

# Redes Neurais

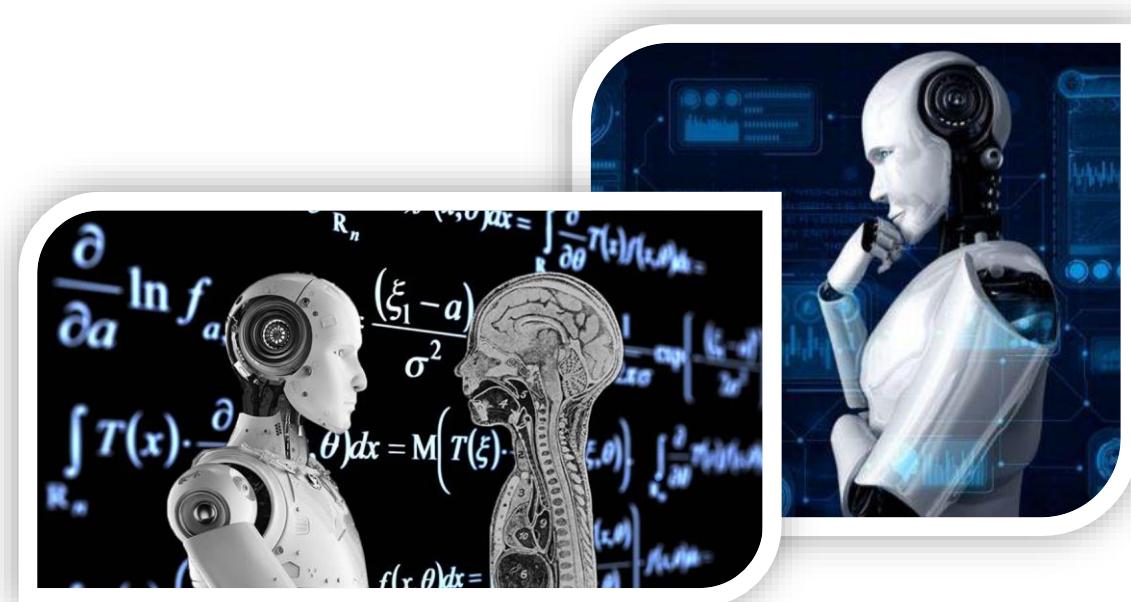
**Disciplina:** Inteligência Computacional (C210A/B)

**Curso:** Engenharia de Computação e Software

Prof<sup>a</sup>. Victoria Dala Pegorara Souto

# REDES NEURAIS

- Promover a compreensão do funcionamento e da aplicação das Redes Neurais na solução de problemas reais.



# MOTIVAÇÃO

- ◎ Como é possível o cérebro processar tantas informações de imagem, sons e outros dados ao mesmo tempo e de forma eficiente?
- ◎ Capacidade de Realizar Tarefas “Simples”:
  - Caminhar;
  - Pegar um objeto;
  - Reconhecer um objeto;
  - ....

Tarefas Difíceis de ensinar  
para um robô!!!



# REDE NEURAL

- ◎ Inspirada na estrutura e no funcionamento do **sistema nervoso**;
- ◎ **Objetivo:** Simular a **capacidade de aprendizado** do cérebro humano na aquisição de conhecimento.



# REDE NEURAL

## → Definição:

Uma rede neural é um **processador** maciçamente paralelamente distribuído constituído de unidades de processamento simples, que tem a propensão natural de **armazenar conhecimento experimental e torná-lo disponível para uso.**

- ✓ As **Redes Neurais Artificiais (RNAs)** são baseadas em **modelos abstratos** de como pensamos que o cérebro (e os neurônios) funciona.



# REDE NEURAL

## ◎ **Principais Características:**

- **Não-linearidade:** consegue mapear fenômenos não-lineares, por exemplo, sinal de voz.
- **Capacidade de aprendizado:** aprendizado a partir de uma sequência de treinamento (exemplos) com saídas desejadas associadas à entradas únicas;
- **Adaptabilidade:** capacidade de adaptar seus pesos sinápticos a partir de modificações do meio-ambiente (novo treinamento);
- **Habilidade de generalização:** após o processo de treinamento da rede, essa é capaz de generalizar o conhecimento adquirido (estimar soluções desconhecidas);
- **Organização de dados:** a rede é capaz de realizar a sua organização interna visando possibilitar o agrupamento de padrões;

# REDE NEURAL

## ◎ **Principais Características:**

- **Informação contextual:** cada neurônio da rede é potencialmente afetado pela atividade de todos os outros neurônios na rede.
- **Tolerância à falhas:** perdas de neurônios não representam degradação significativas na respostas de forma global (hardware);
- **Implementação em larga escala:** adequada para implementação utilizando tecnologia de integração em larga escala (natureza paralela);
- **Analogia neurobiológica:** motivado pela semelhança com o cérebro (prova viva de que o processamento paralelo é rápido e poderoso);  
**Facilidade de prototipagem:** pode ser implementada facilmente em software e hardware (processo de execução).

# RESUMO HISTÓRICO

1943

- Primeira publicação relacionada à neurocomputação (McCulloch & Pitts);
- Primeiro modelamento matemático inspirado em um neurônio biológico;

1949

- Primeiro método de treinamento para redes neurais denominado regra de aprendizado de Hebb;

1958

- Primeiro neurocomputador (Perceptron Mark I) desenvolvido por Frank Rosenblatt;

1960

- Widrow & Hoff desenvolveram um tipo de rede denominada *Adaline* cujo aprendizado é fundamentado na chamada regra Delta;

1969

- Minsky & Papert (em “Perceptrons”) apresentaram a limitação das redes neurais, como *Perceptron* e *Adaline*, com problemas não linearmente separáveis;
- Congelamento da área de RN;

# RESUMO HISTÓRICO

1982

- Hopfield propôs as redes recorrentes baseadas em funções de energia;
- Retomada da área de RN;

1986

- Publicação do livro *Parallel distributed processing* [Rumelhart *et al.*], propondo um algoritmo que permitia ajustar os pesos em uma rede com mais de uma camada (**backpropagation**); Resolveram problemas não linearmente separáveis;

1992

- Riedmiller & Braun propuseram o método Resillient Propagation (Rprop), permitindo a atuação individual em pesos sinápticos e agilizando a convergência.
- Igel & Hüskens nomearam e propuseram variações.

1994

- Algoritmos de aprendizado baseados no método Levenberg-Marquardt que permite incrementar a eficiência do treinamento de redes neurais artificiais [Hagan & Menhaj];

1998

- Apresentação das máquinas de vetores suporte (*support vector machines - SVM*), modelos que podem ser utilizados em classificação de padrões, regressão e agrupamento [Vapnik];

# RESUMO HISTÓRICO

2002

- Ken Stanley (Universidade do Texas) propôs o NEAT (*NeuroEvolution of Augmenting Topologies*), um algoritmo genético (GA) para a geração de redes neurais artificiais evolucionárias (“complexificação” de topologias).

2003

- Implementação de circuitos integrados neurais com diversas configurações de topologia [Beiu *et al*];

2012

- Krizhevsky et. al. usaram redes convolutivas e conseguiram reduzir em 50% a taxa de erro no reconhecimento de objetos, precipitando a rápida adoção do aprendizado profundo pela comunidade de visão computacional.

2014

- Os pesquisadores do Facebook publicam seu trabalho DeepFace, um sistema que usa redes neurais capaz de identificar rostos com precisão de 97,35%.

2016

- O programa AlphaGo do Google se torna o primeiro a vencer um jogador humano profissional balanceado usando uma combinação de técnicas de aprendizado de máquina e busca de árvores.

# ÁREAS DE APLICAÇÃO DE RNAs

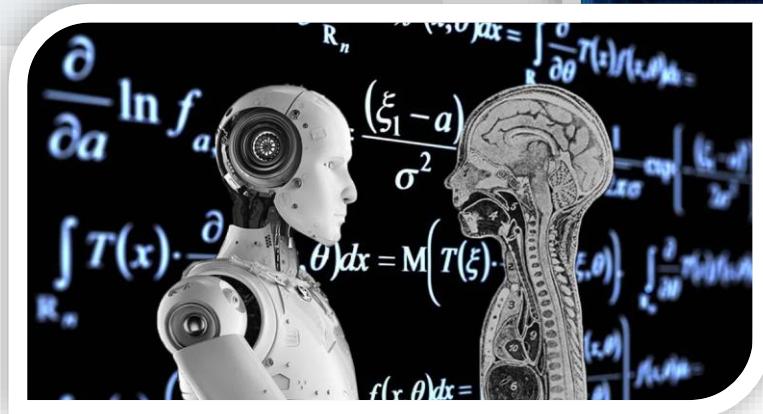
- ◎ **Aproximador universal de funções:** tem como objetivo mapear o relacionamento funcional entre as variáveis (reais) de um sistema a partir de um conjunto de valores conhecidos;
  - ✓ Mapeamento de processos diversos
- ◎ **Controle de processos:** tem como objetivo identificar ações de controle que permitam o alcance dos requisitos de um processo;
  - ✓ Robótica, aeronaves, elevadores, etc.
- ◎ **Reconhecimento / classificação de padrões:** tem como objetivo associar um padrão de entrada para uma das classes previamente definidas;
  - ✓ Reconhecimento de imagens, voz, escrita, etc.
- ◎ **Otimização de sistemas:** tem como objetivo minimizar ou maximizar uma função de custo (objetivo);
  - ✓ Otimização restrita, programação dinâmica, otimização combinatorial

# ÁREAS DE APLICAÇÃO DE RNAs

- ◎ **Agrupamento de dados (clusterização):** tem como objetivo identificar e detectar semelhanças e particularidades entre os diversos padrões de entrada para efetuar o agrupamento;
  - ✓ Identificação automática de classes
- ◎ **Sistemas de previsão:** tem como objetivo estimar valores futuros de um processo levando-se em consideração diversas medidas prévias observadas em seu domínio;
  - ✓ Previsão de séries temporais, mercados financeiros, previsões climáticas
- ◎ **Memórias associativas:** tem como objetivo recuperar padrões corretos mesmo se os seus elementos constituintes forem apresentados de forma incerta ou imprecisa;
  - ✓ Processamento de imagens, transmissão de sinais, etc.

# SISTEMA NERVO

- As **Redes Neurais Artificiais (RNAs)** são baseadas em **modelos abstratos** de como pensamos que o cérebro (e os neurônios) funciona.



# SISTEMA NERVOSO

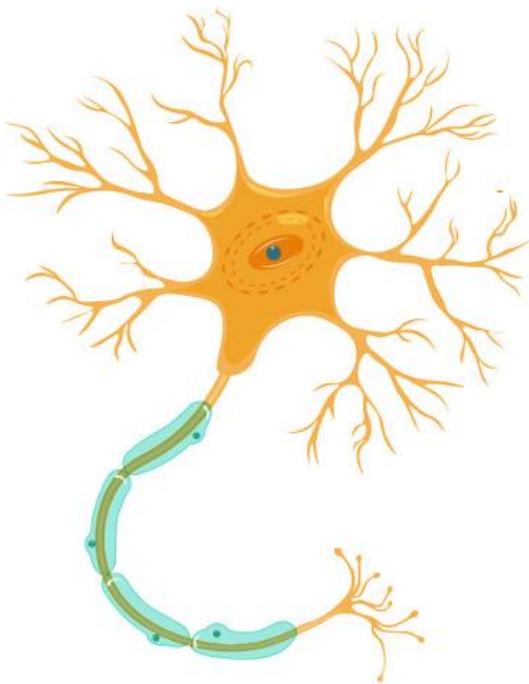
- ◎ **Conjunto complexo de células** que determinam o funcionamento e o comportamento de seres vivos.

**Unidade Fundamental do Sistema Nervoso:**

→ Neurônio

- ◎ **Neurônio** → Se distingue das outras células por apresentar **excitabilidade**, que lhe permite **responder a estímulos externos e internos**. Isso possibilita a transmissão de impulsos nervosos a outros neurônios e a células musculares e glandulares.

**Componentes de um Neurônio:** Dendritos, Corpo Celular (Soma) e Axônio.

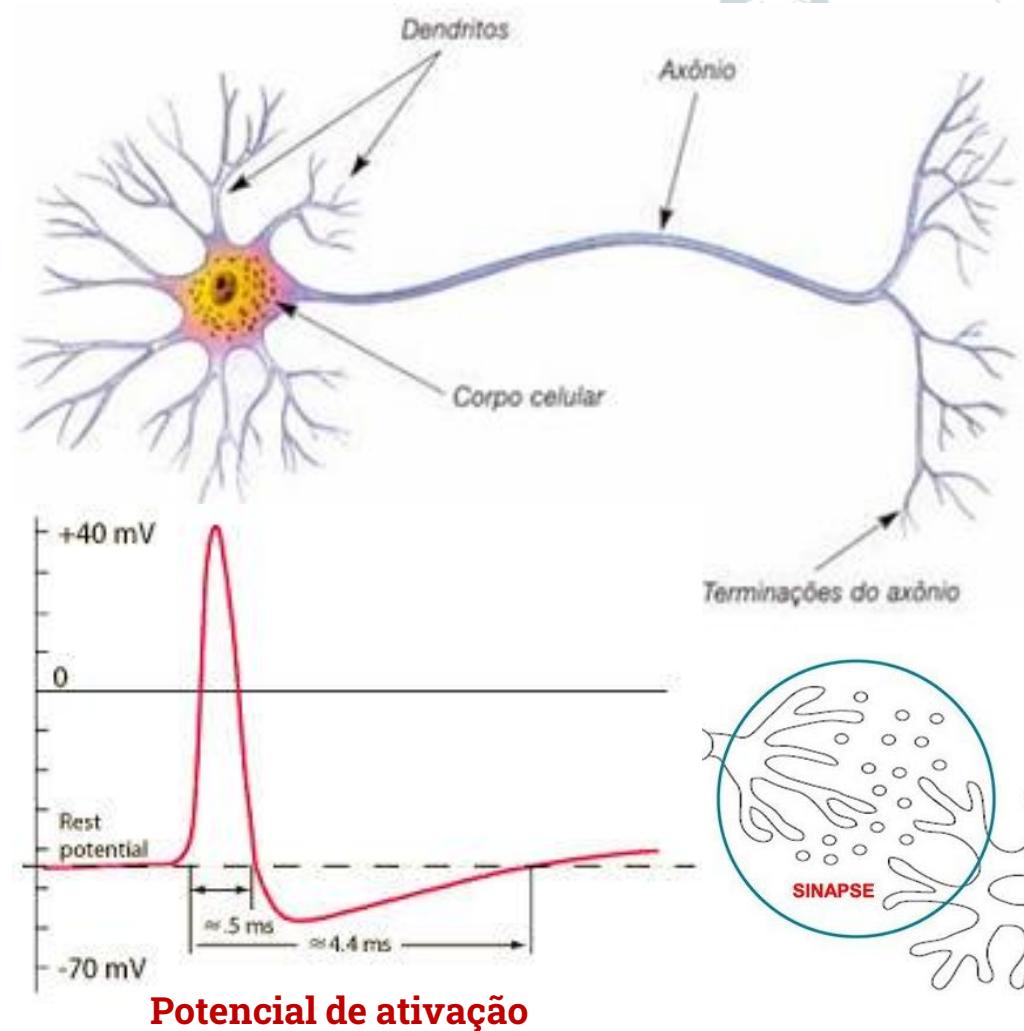


# SISTEMA NERVOSO

## Componentes de um Neurônio Biológico:

- **Dendritos:** Prolongamentos dos neurônios especializados na **recepção de estímulos nervosos provenientes de outros neurônios ou do ambiente.**
- Esse estímulos são então transmitidos para o corpo celular ou soma.
- De acordo com a intensidade e frequência dos estímulos **recebidos**, o corpo celular gera um novo impulso, que é enviado para o **Axônio**.

- Unidade de processamento de informação que é fundamental para a operação de uma rede neural.

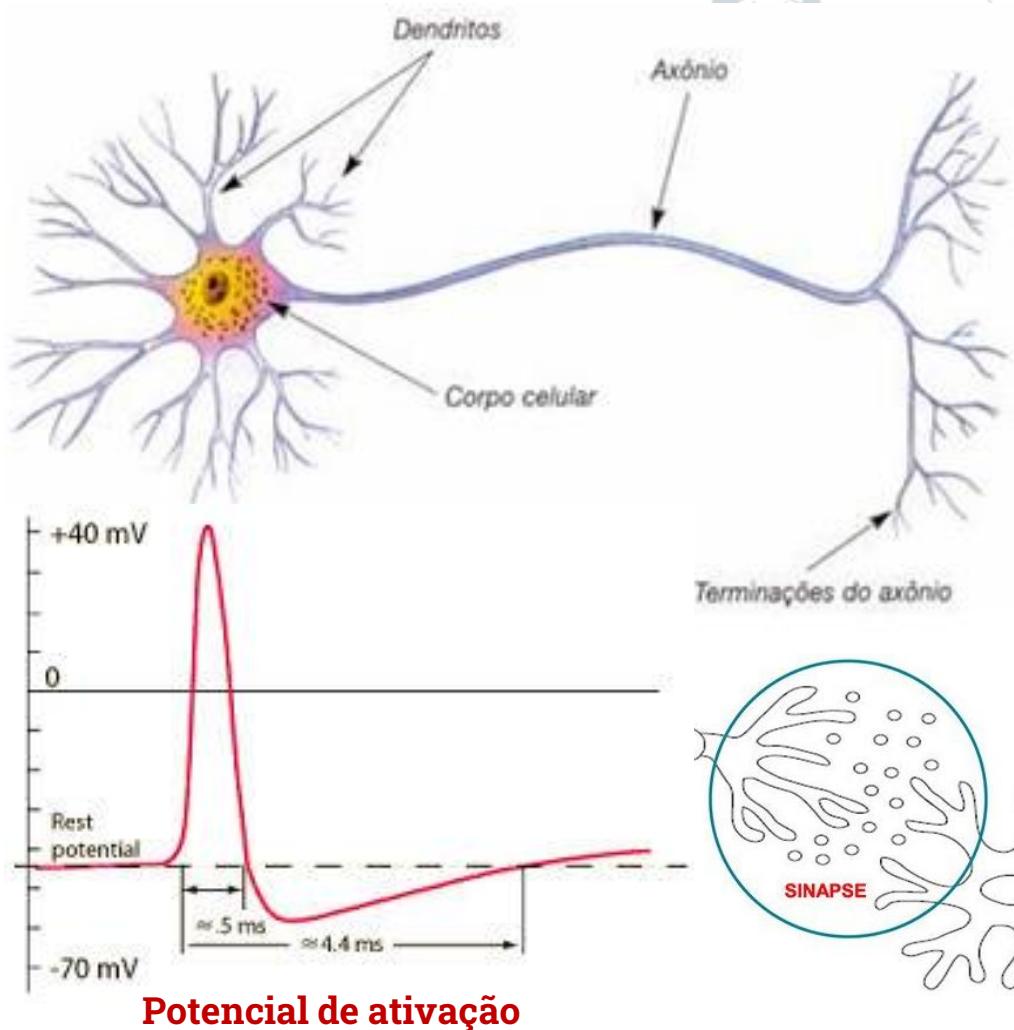


# SISTEMA NERVOSO

## ◎ Componentes de um Neurônio Biológico:

- **Axônio:** Prolongamento dos neurônios, responsável pela **condução dos impulsos elétricos** produzidos no **corpo celular** até outro local mais distante (usualmente até outros neurônios).
- O sinal do neurônio flui então da **esquerda para a direita**, ou seja, dos **dendritos para o corpo celular e em seguida para o axônio**.
- O contato entre a terminação de uma axônio e o dentrito de outro neurônio é denominado **Sinapse**.

- Unidade de processamento de informação que é fundamental para a operação de uma rede neural.

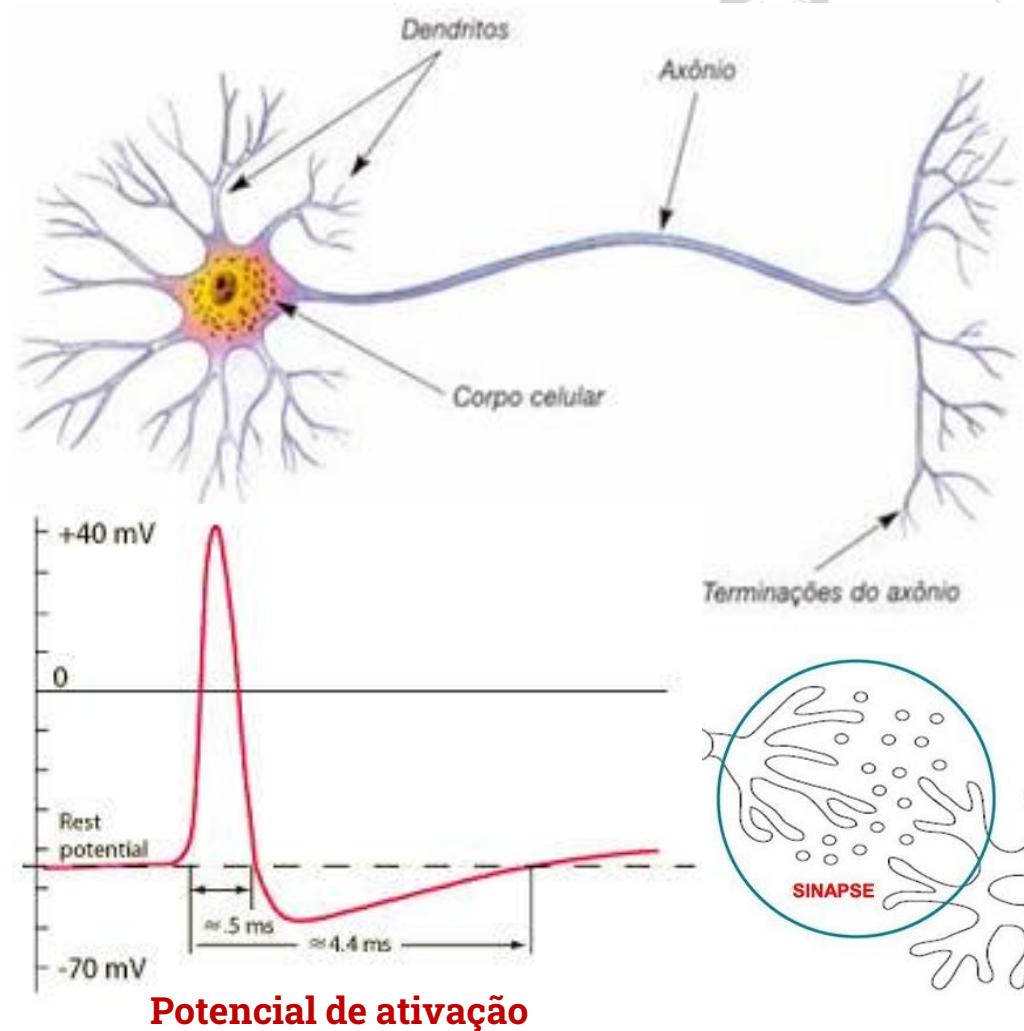


# SISTEMA NERVOSO

## ◎ Componentes de um Neurônio Biológico:

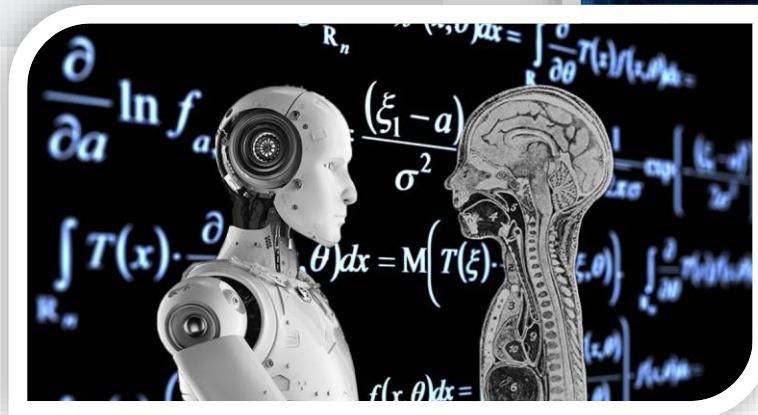
- **Sinapse:** As sinapses são, portanto, as **unidades que medeiam as interações entre os neurônios**, e podem ser **excitatórias ou inibitórias**.
- O cérebro humano possui um grande número de neurônios, da ordem de 10 a 500 milhões. De acordo com estimativas, eles encontram-se organizados em aproximadamente 1000 módulos principais, cada um com 500 redes neurais.
- Além disso, cada neurônio pode estar conectado a centenas ou até mesmo milhares de outros neurônios. **Essas redes biológicas trabalham de forma massivamente paralela**, provendo uma grande rapidez de processamento.

- Unidade de processamento de informação que é fundamental para a operação de uma rede neural.



# COMPONENTES BÁSICOS DAS RNAs

- As **Redes Neurais Artificiais (RNAs)** são baseadas em **modelos abstratos** de como pensamos que o cérebro (e os neurônios) funciona.



# COMPONENTES BÁSICOS DAS RNAs

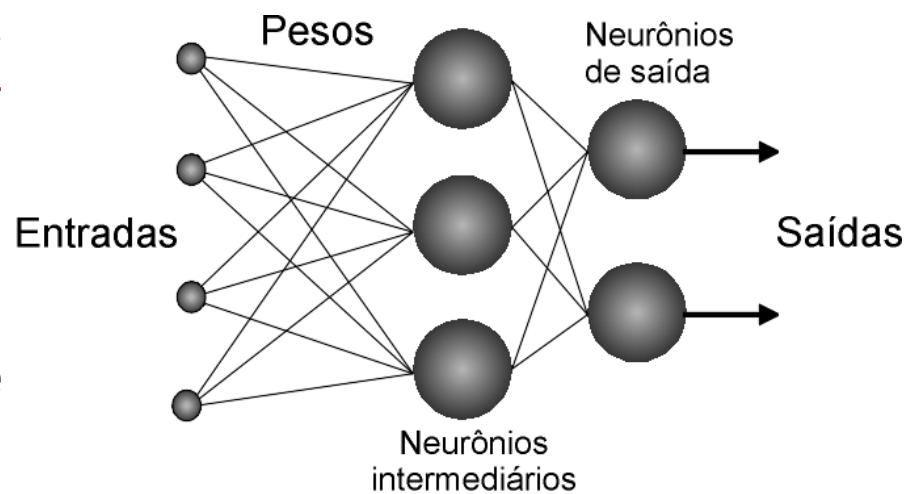
- RNAs são sistemas computacionais distribuídos compostos de unidades de processamento simples (**Neurônios Artificiais**), densamente interconectadas.

- Neurônios Artificiais →** Computam funções matemáticas.

- Os neurônios são dispostos em uma ou mais camadas interligadas por um grande número de conexões, geralmente unidirecionais.
- Essas conexões, que simulam as **sinapses biológicas**, possuem **pesos** associados, que **ponderam a entrada recebida por cada neurônio da rede**.

- Pesos positivos →** Conexão Excitatória
- Pesos negativos →** Conexão Inibitória.

Os pesos têm seus valores ajustados em um **processo de aprendizado** e **codificam o conhecimento adquirido pela rede**.



# COMPONENTES BÁSICOS DAS RNAs

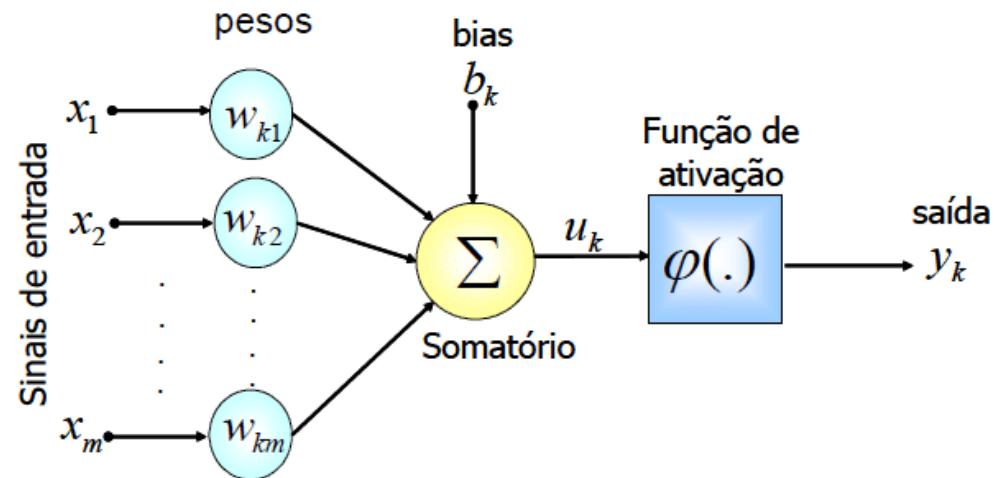
- Uma RNA é portanto caracterizada por **dois aspectos básicos**:

## **ARQUITETURA + APRENDIZADO**

- **Arquitetura** → Está relacionada ao tipo e número de unidades de processamento e à forma como os neurônios estão conectados;
- **Aprendizado** → Está relacionado às regras utilizadas para o ajuste dos pesos da rede e que informação é utilizada pelas regras.

# COMPONENTES BÁSICOS DAS RNAs

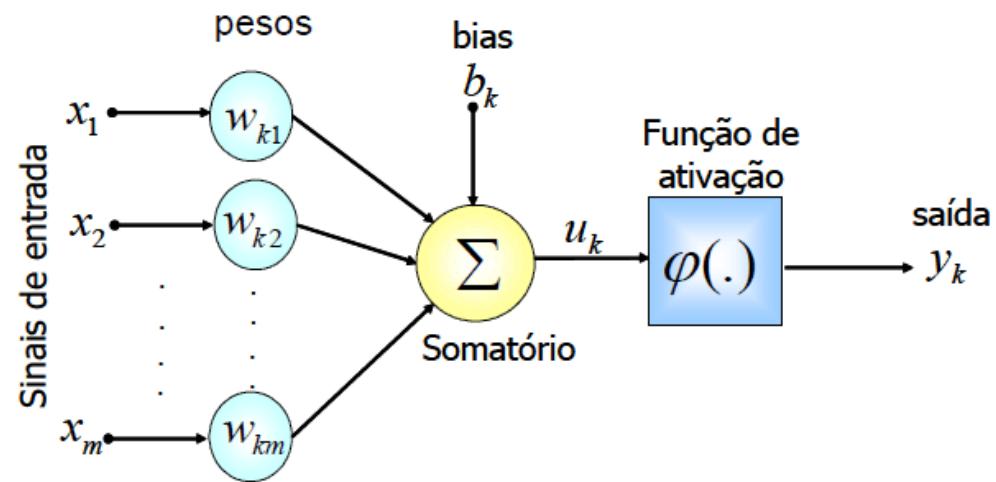
- A arquitetura de uma RNA define a forma como os seus neurônios estão arranjados uns em relação aos outros;



- Cada **terminal de entrada** do neurônio, simulando os dendritos, recebe um valor. Os valores recebidos são **ponderados e combinados por uma função matemática (Função de Ativação)**  $\varphi(\cdot)$ , equivalendo ao processamento realizado pela soma.  
A **saída da função** é a resposta do neurônio para a entrada.

# COMPONENTES BÁSICOS DAS RNAs

- A arquitetura de uma RNA define a forma como os seus neurônios estão arranjados uns em relação aos outros;

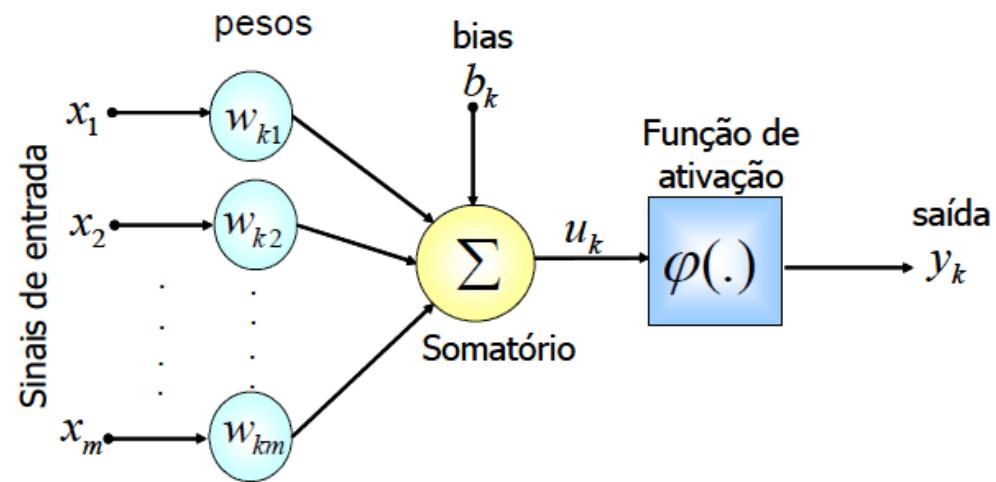


- Suponha um objeto  $x$  com  $d$  atributos representado como  $x = [x_1, x_2, \dots, x_d]^T$  e um neurônio com  $d$  terminais de entrada cujos pesos são  $w_1, w_2, \dots, w_d$ , que podem ser representados como  $w = [w_1, w_2, \dots, w_d]$ . A entrada total recebida pelo neurônio,  $u$ , pode ser definida como:

$$\rightarrow u = \sum_{j=1}^d x_j w_j$$

# COMPONENTES BÁSICOS DAS RNAs

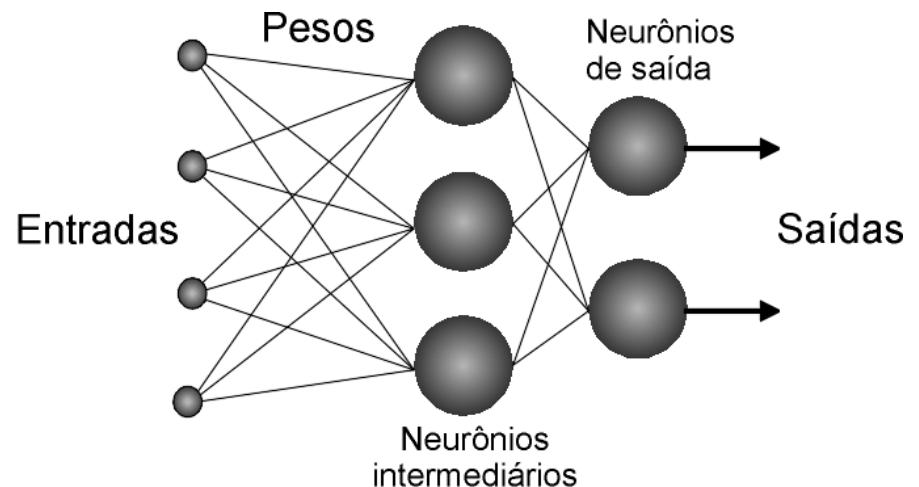
- A arquitetura de uma RNA define a forma como os seus neurônios estão arranjados uns em relação aos outros;



- Os neurônios podem apresentar **conexões de entrada negativas ( $w_j < 0$ ) ou positivas ( $w_j > 0$ )**.
- Um valor de **peso igual a zero** equivale a **ausência da conexão associada**.
- A saída de um neurônio é definida por meio da aplicação de uma **função de ativação à entrada total**.

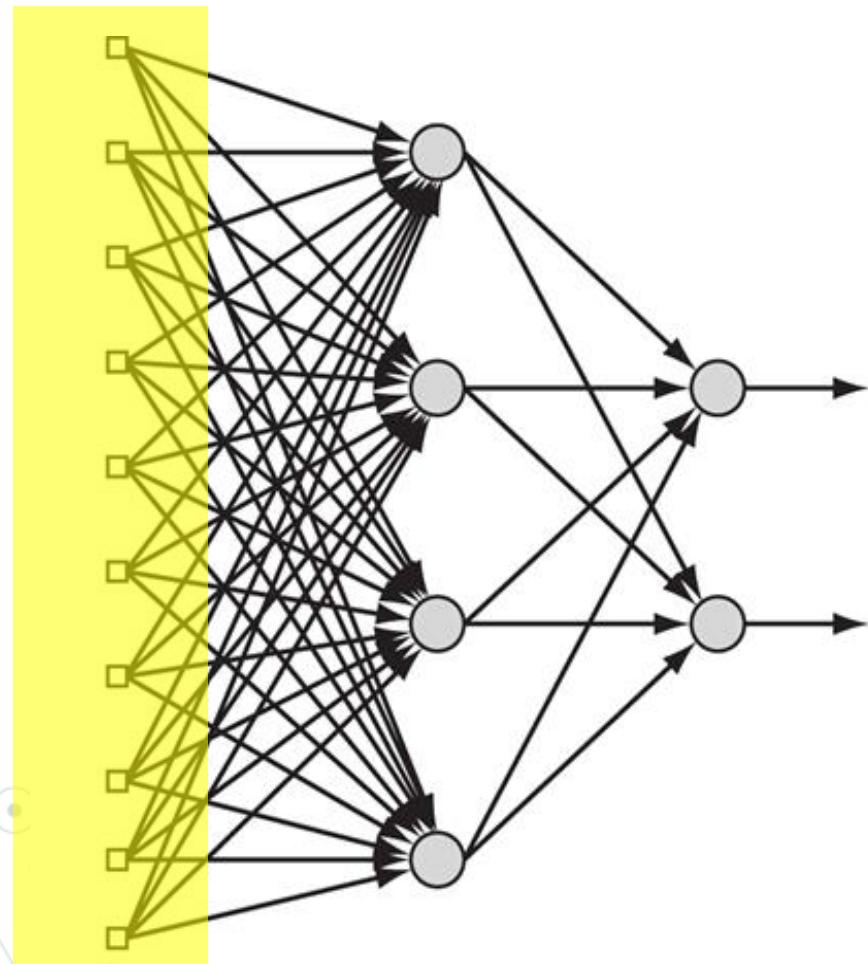
# COMPONENTES BÁSICOS DAS RNAs

- Em uma RNA, os neurônios podem estar dispostos em uma ou mais camadas.
- Quando duas ou mais camadas são utilizadas, um neurônio pode receber em seus terminais de entrada valores de saída de neurônios da camada anterior e/ou enviar seu valor de saída para terminais de entrada de neurônios da camada seguinte.
- Uma rede neural de **múltiplas camadas** pode ser dividida em três partes básicas:
  - Camada de entrada;
  - Camada escondida;
  - Camada de saída.



# COMPONENTES BÁSICOS DAS RNAs

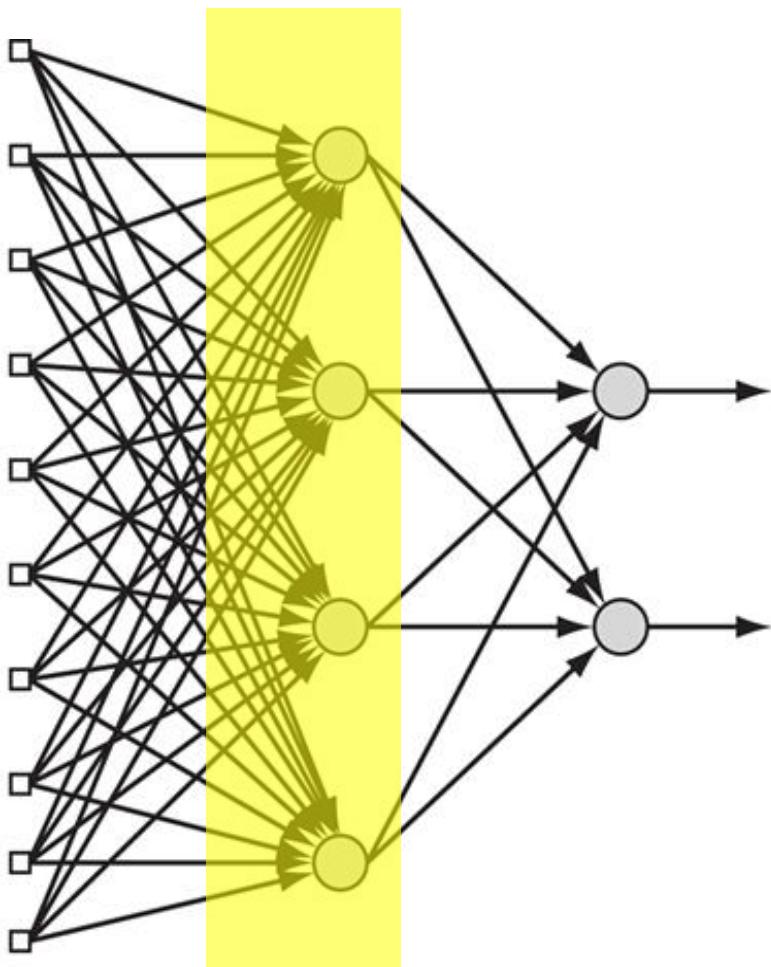
## Camada de entrada



→ Responsável pelo recebimento de informações (dados) do meio externo;

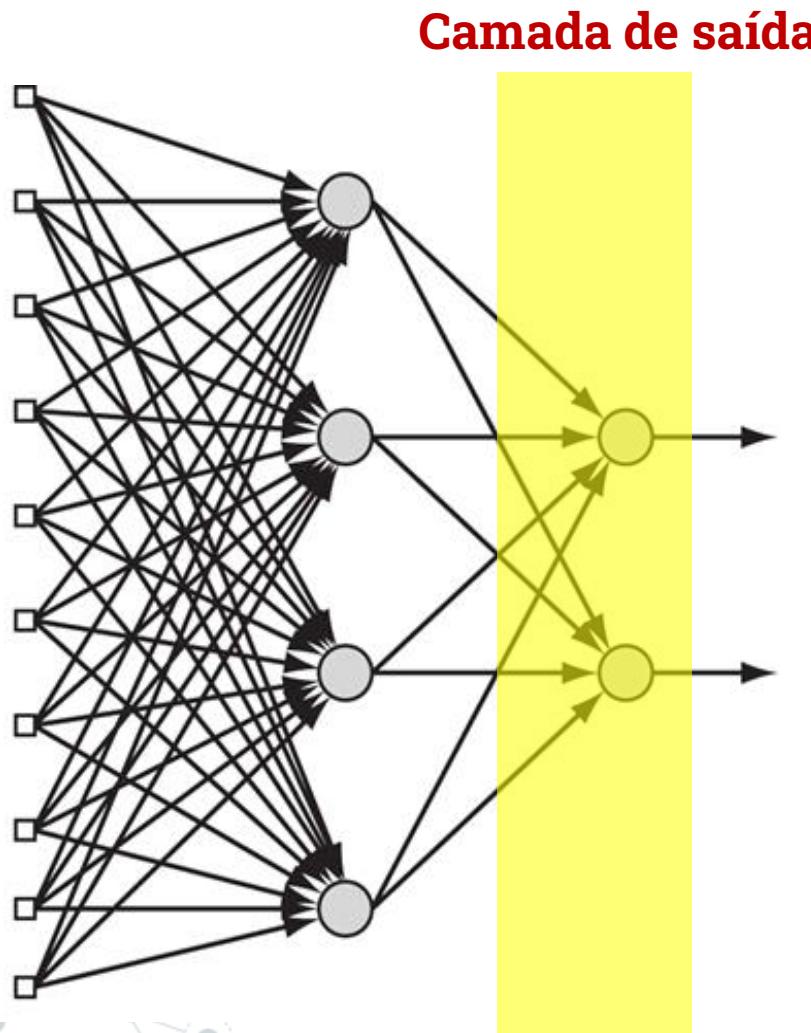
# COMPONENTES BÁSICOS DAS RNAs

## Camada escondida ou intermediária



→ Compostas por neurônios que possuem a responsabilidade de extrair as características associadas ao processo ou sistema a ser inferido;

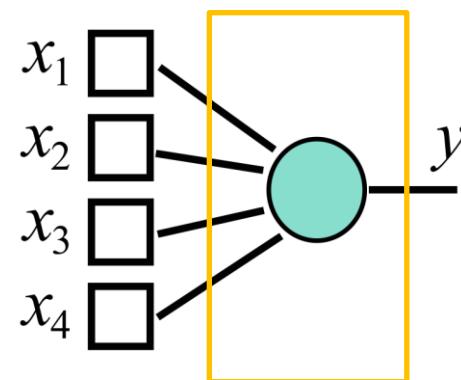
# COMPONENTES BÁSICOS DAS RNAs



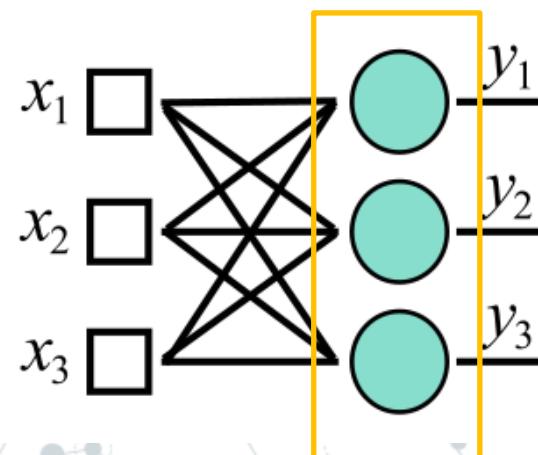
→ Também constituída de neurônios sendo responsável pela produção e apresentação dos resultados finais da rede;

# COMPONENTES BÁSICOS DAS RNAs

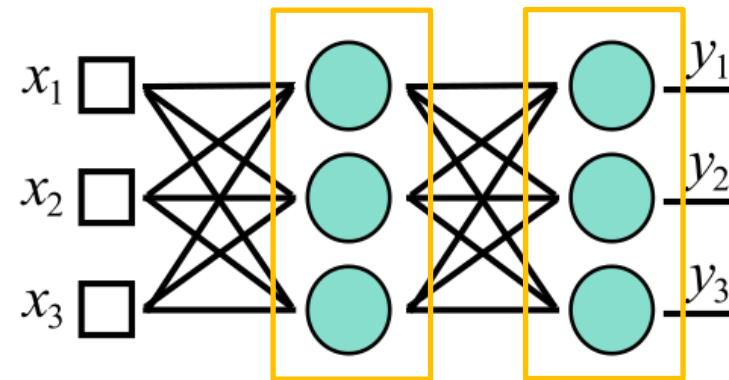
- **Exemplos:**



- ✓ Uma Camada de Saída com 1 Neurônio



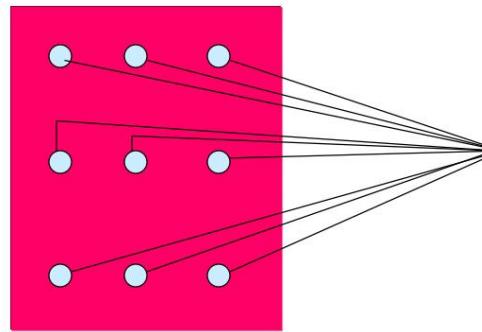
- ✓ Uma Camada de Saída com 3 Neurônios



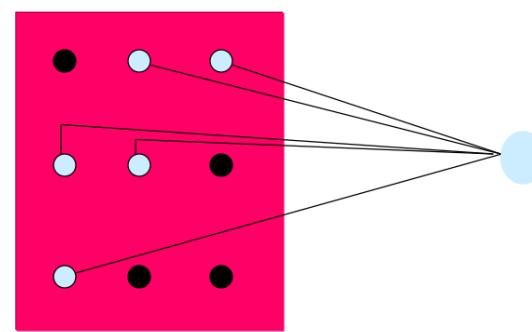
- ✓ Uma Camada Escondida com 3 Neurônios.
- ✓ Uma Camada de Saída com 3 Neurônios.

# COMPONENTES BÁSICOS DAS RNAs

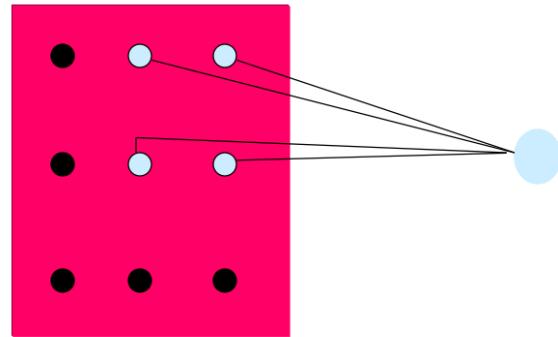
- Em uma rede multicamadas, as **conexões entre os neurônios podem apresentar diferentes padrões de conexão**. De acordo com esses padrões, a rede pode ser classificada em:
  - Completamente Conectada:** Quando os neurônios da rede estão conectados a todos os neurônios da camada anterior e/ou seguite.
  - Parcialmente Conectada:** Quando os neurônios estão conectados a apenas alguns dos neurônios da camada anterior e/ou seguinte.
  - Localmente Conectada:** São redes parcialmente conectadas, em que os neurônios conectados a um neurônio se encontram em uma região bem definida.



**Completamente Conectada**



**Parcialmente Conectada**



**Localmente Conectada**

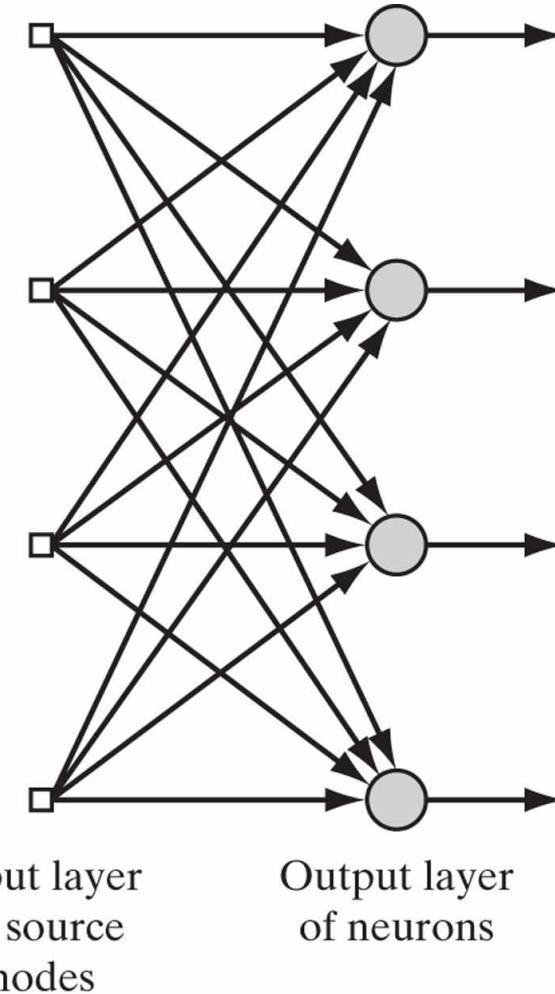
# COMPONENTES BÁSICOS DAS RNAs

- Além do grau de conectividade, as RNAs podem **apresentar ou não** conexões de retroalimentação, ou feedback.
- A informação em uma Rede Neural geralmente **flui** da camada de entrada da rede para os neurônios da camada de saída.
- Para redes multicamadas esse fluxo ocorre camada a camada.**
- As **conexões de retroalimentação** permitem que um neurônio receba em seus terminais de entrada a saída de um neurônio da **mesma camada** ou de uma **camada posterior**.
- O neurônio pode inclusive **receber sua própria saída** em um de seus terminais de entrada.
  - Exemplo de Aplicação:** Processamento de língua natural e o controle de braços robóticos.

# COMPONENTES BÁSICOS DAS RNAs

- Classificação das RNAs:
- **Rede Feedforward com Camada Única:**

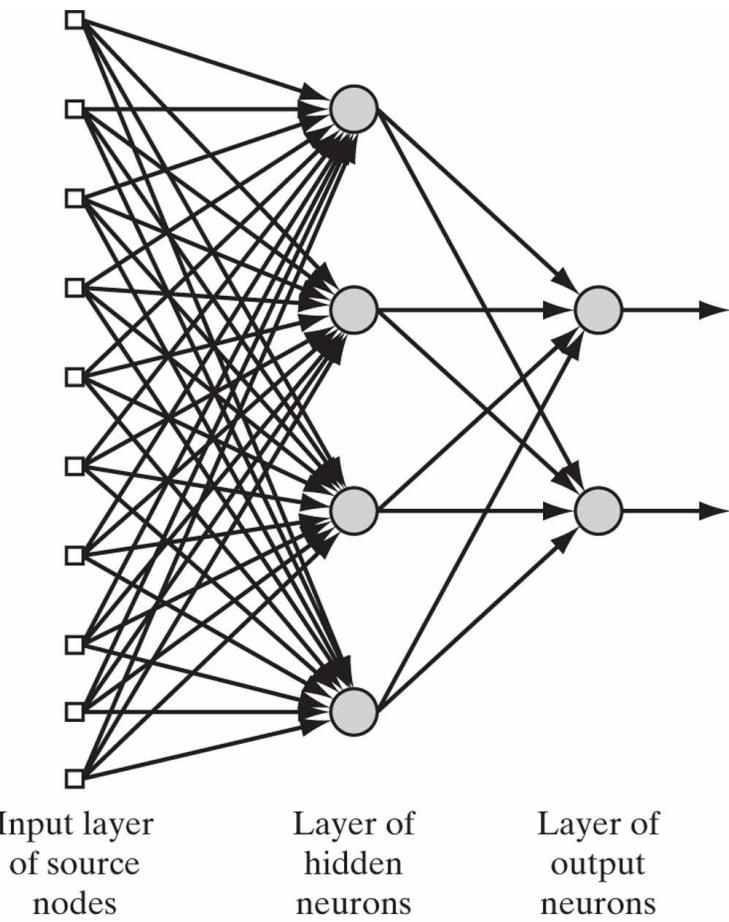
- Uma camada de entrada de nós-fonte que se projeta sobre uma camada de saída de neurônios.



# COMPONENTES BÁSICOS DAS RNAs

- Classificação das RNAs:
- **Rede Feedforward com Múltiplas Camadas:**

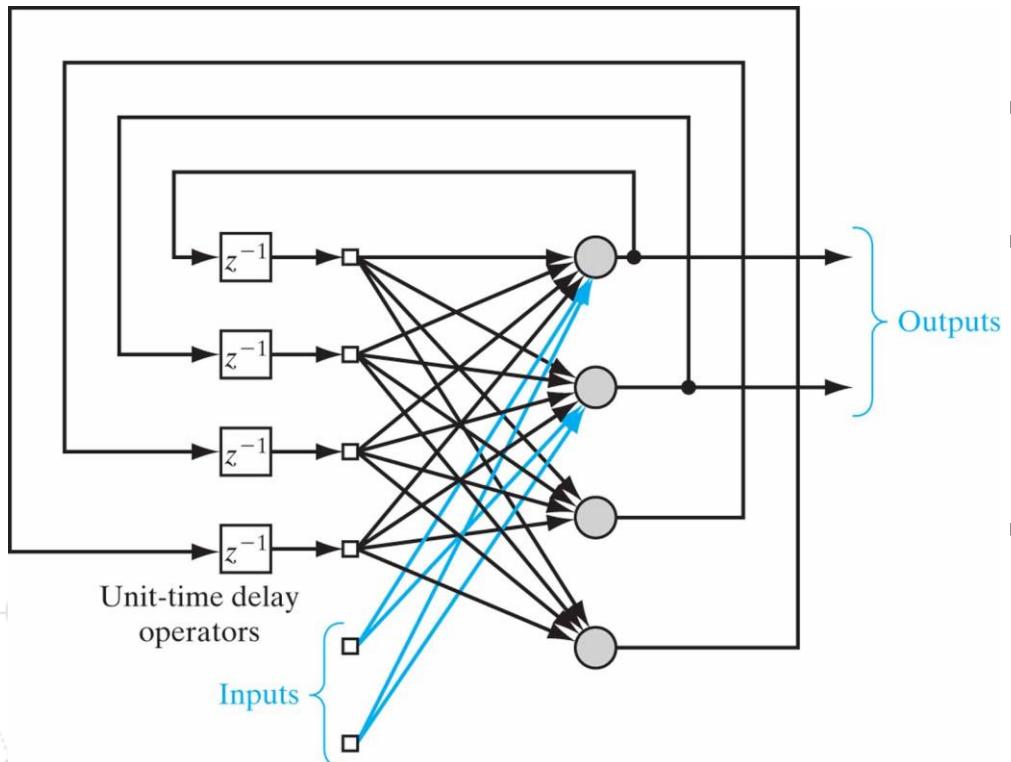
- Os nós-fonte da camada de entrada da rede fornecem os respectivos elementos do padrão de ativação, que constituem os sinais de entrada aplicados aos neurônios na segunda camada.



# COMPONENTES BÁSICOS DAS RNAs

- Classificação das RNAs:

- **Redes Recorrentes ou Realimentadas:**

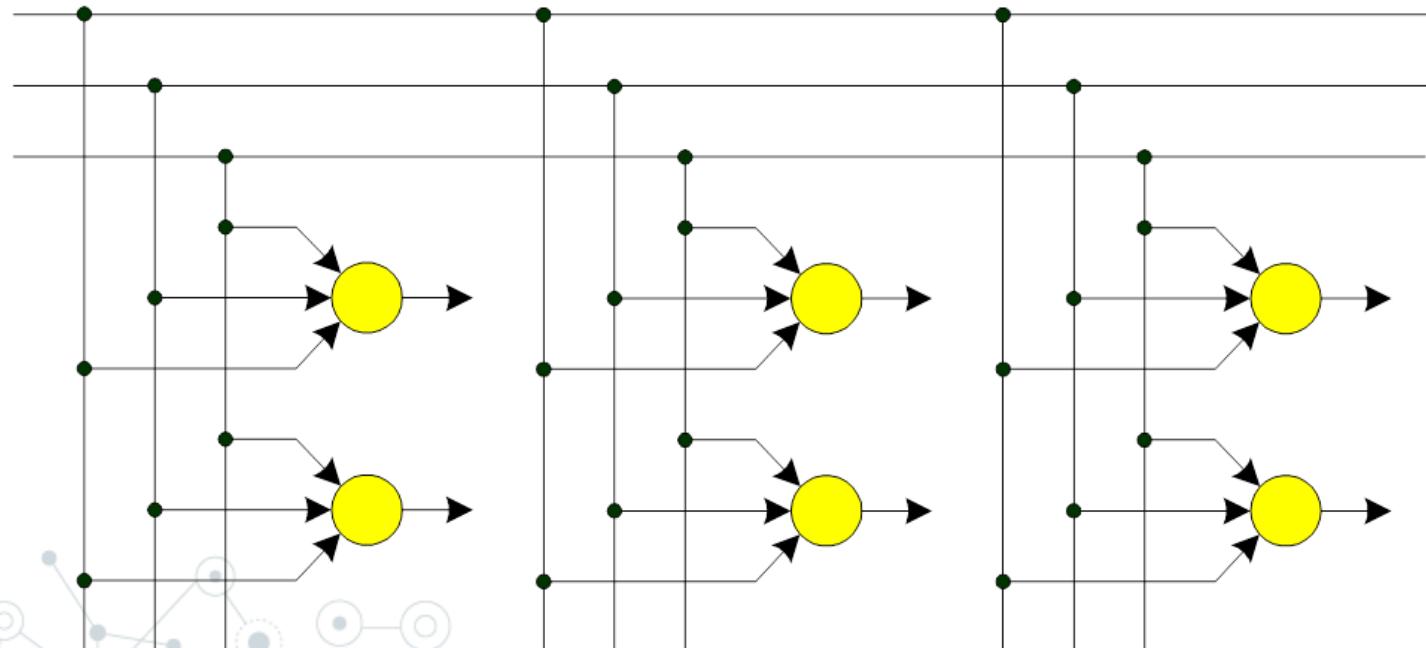


- Difere da rede neural alimentada adiante (*feedforward*) por possuir pelo menos um laço de realimentação.
- Recebem informações de entradas anteriores para influenciar a entrada e a saída atuais.
- As redes com retropropagação, conhecidas como redes recorrentes, são indicadas para aplicações em que é necessário **processar informações sequenciais e na simulação de sistemas dinâmicos**.
- **Exemplo de Aplicação:** Tradução de idiomas, processamento de linguagem natural, reconhecimento de fala e legendagem de imagens.

# COMPONENTES BÁSICOS DAS RNAs

- Classificação das RNAs:

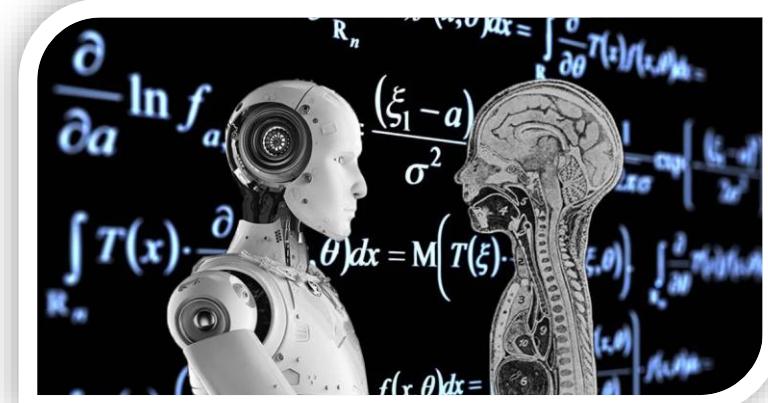
- Redes em Estrutura Reticulada:



- Tem como principal característica a **disposição espacial dos neurônios**.
- Consideram a disposição espacial dos neurônios com o propósito da **extração de características do sistema**, ou seja, sua localização espacial serve para ajuste de seus pesos e limiares.
- São empregadas em problemas de agrupamento, reconhecimento de padrões, otimização de sistemas.
- Muito aplicada em problemas de agrupamento (mapas de Kohonen).

# NEURÔNIO ARTIFICIAL

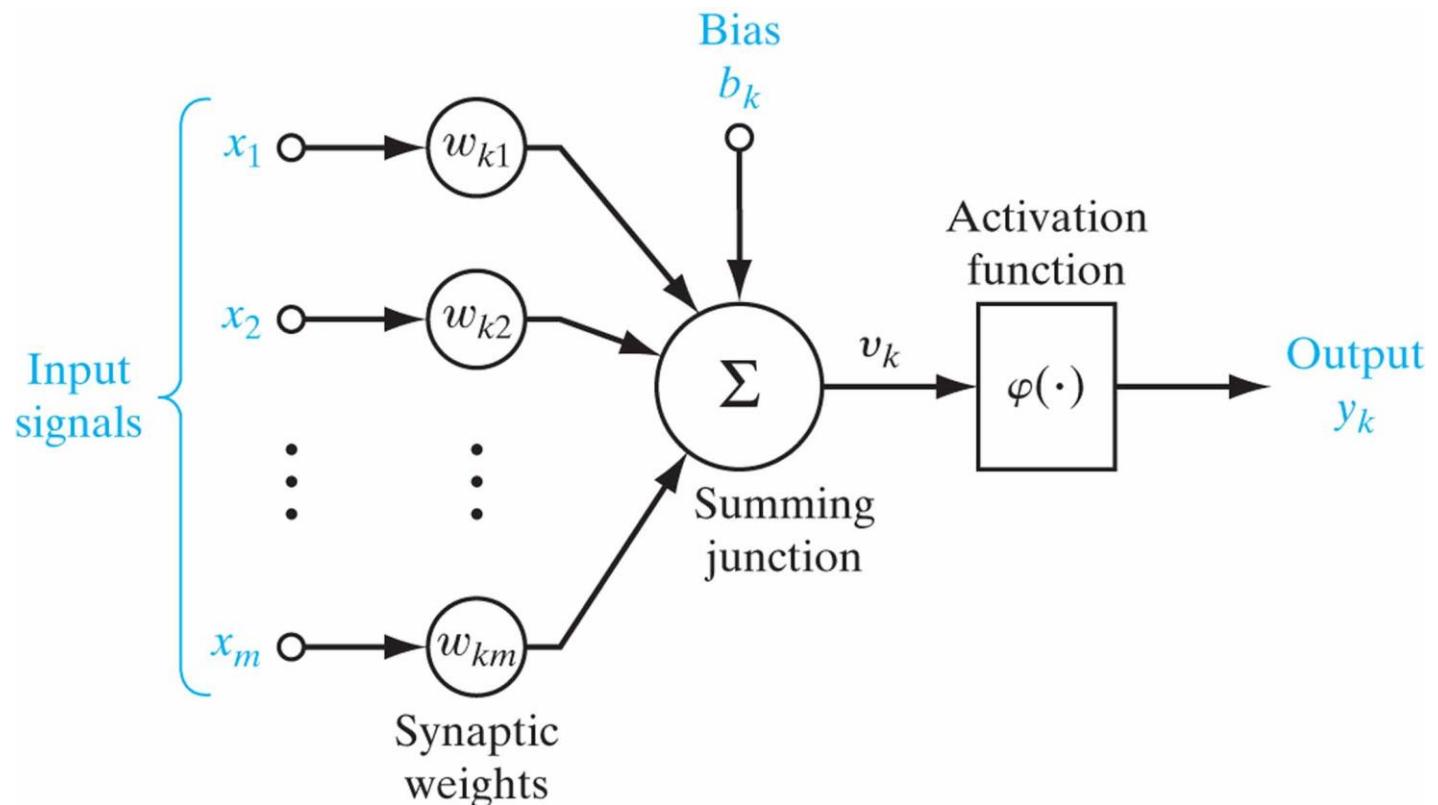
- Unidade fundamental das Redes Neurais Artificiais.


$$\frac{\partial}{\partial a} \ln f_a(x, \theta) dx = \int_{\mathbb{R}_n} \frac{\partial}{\partial \theta} T(x) f_a(x, \theta) dx$$
$$\int_{\mathbb{R}_n} T(x) \cdot \frac{\partial}{\partial \theta} f_a(x, \theta) dx = M(T(\xi), \theta)$$
$$f_a(x, \theta) dx =$$



# NEURÔNIO ARTIFICIAL

- Modelo matemático não-linear de um neurônio  $k$ . (McCulloch & Pitts):



# NEURÔNIO ARTIFICIAL

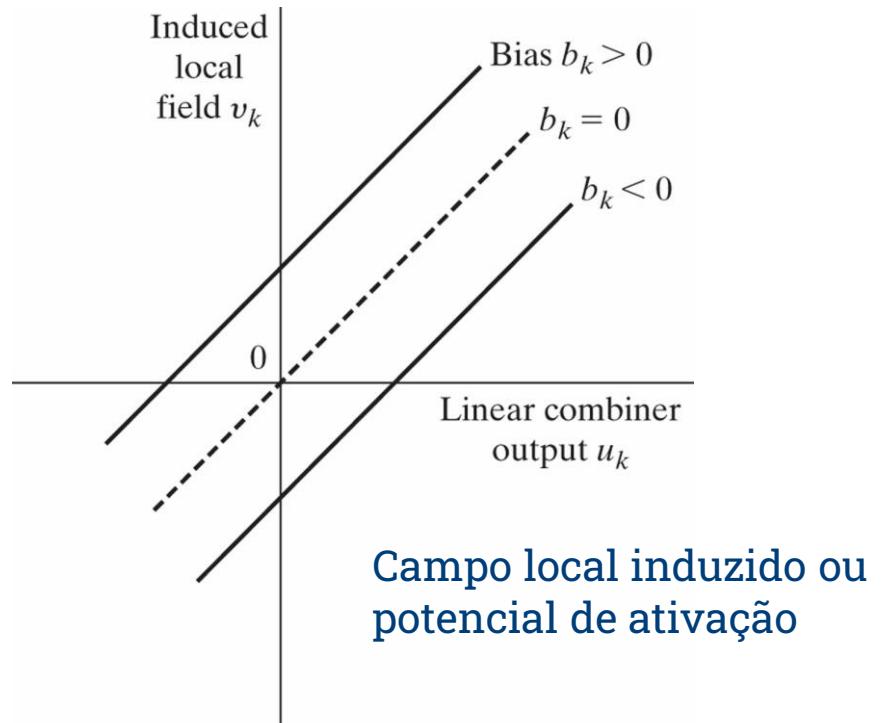
## ◎ Elementos Básicos de um Neurônio Artificial:

- **Sinais de entrada  $\{x_1, x_2, \dots, x_m\}$ :** sinais ou medidas do meio externo que representam os valores das variáveis;
- **Pesos sinápticos  $\{w_{k1}, w_{k2}, \dots, w_{km}\}$ :** valores que servirão para ponderar cada uma das variáveis de entrada (indicam a relevância de cada entrada na rede);
- **Combinador linear  $\{\Sigma\}$ :** sua função é agregar todos os sinais de entrada que foram ponderados pelos respectivos pesos sinápticos a fim de gerar um valor de potencial de ativação;

# NEURÔNIO ARTIFICIAL

## ◎ Elementos Básicos de um Neurônio Artificial:

- **Limiar de ativação  $\{\theta\}$  ou  $\{b_k\}$ :** variável que especifica qual será o patamar apropriado para que o resultado produzido pelo combinador linear possa gerar um valor de disparo em direção a saída do neurônio;



# NEURÔNIO ARTIFICIAL

## ◎ Elementos Básicos de um Neurônio Artificial:

- **Potencial de ativação  $\{v_k\}$ :** resultado produzido pela diferença do valor produzido entre o combinador linear e o limiar de ativação. Se tal valor é positivo, ou seja, **se  $u_k + b_k \geq 0$  então o neurônio produz um potencial excitatório; caso contrário, o potencial será inibitório;**

$$u_k = \sum_{j=1}^m w_{kj} \cdot x_j$$

$$v_k = u_k + b_k$$

# NEURÔNIO ARTIFICIAL

## ◎ Elementos Básicos de um Neurônio Artificial:

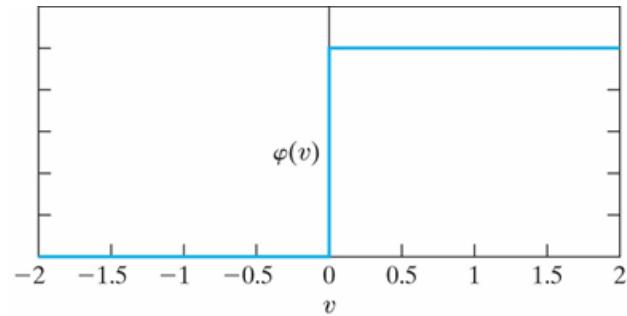
- **Função de ativação  $\{\varphi\}$ :** seu objetivo é limitar a saída do neurônio dentro de um intervalo de valores razoáveis a serem assumidos pela sua própria imagem funcional;
- **Sinal de saída  $\{y_k\}$ :** valor final produzido pelo neurônio em relação a um determinado conjunto de sinais de entrada, podendo ser também utilizado por outros neurônios que estão sequencialmente interligados.:

$$y_k = \varphi(u_k + b_k) = \varphi(v_k)$$

# NEURÔNIO ARTIFICIAL

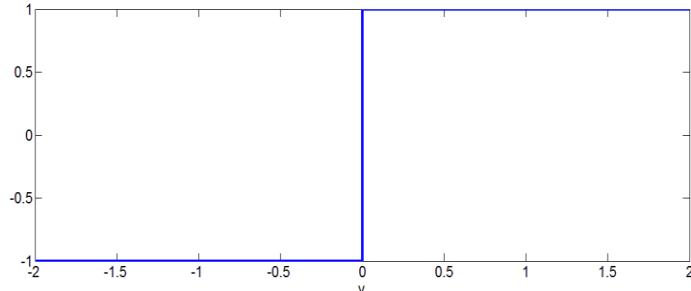
## ◎ Tipos de Funções de Ativação:

### ○ Função de limiar ou função de *heaviside*:



$$\varphi(v) = \begin{cases} 1 & \text{se } v \geq 0 \\ 0 & \text{se } v < 0 \end{cases}$$

### ○ Função de *heaviside simétrica*:



$$\varphi(v) = \begin{cases} +1 & \text{se } v > 0 \\ 0 & \text{se } v = 0 \\ -1 & \text{se } v < 0 \end{cases}$$

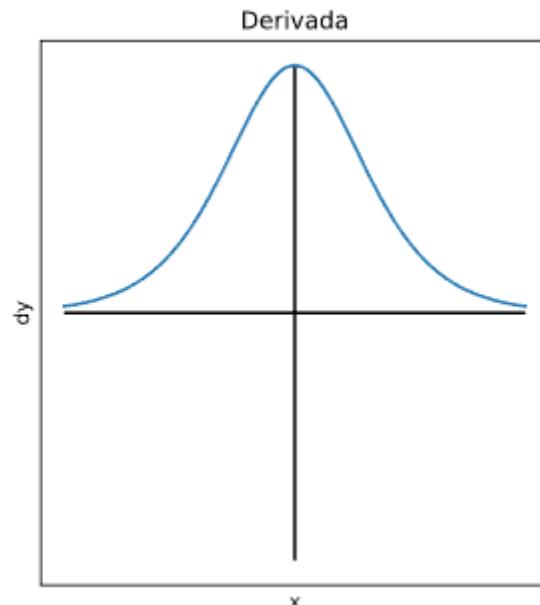
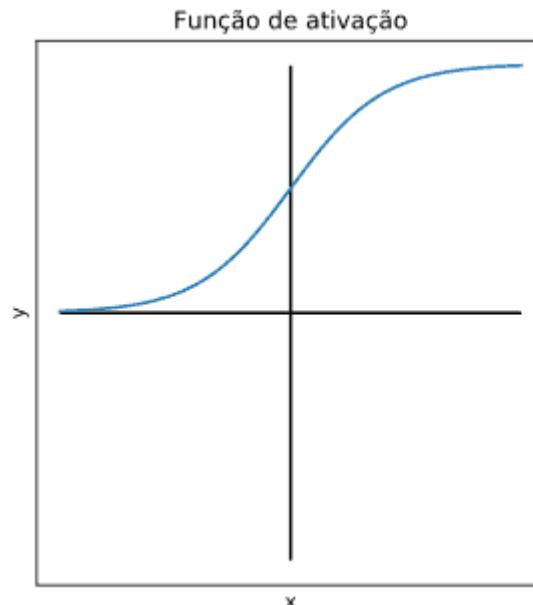
# NEURÔNIO ARTIFICIAL

## ◎ Tipos de Funções de Ativação:

### ○ Função de ativação logística (sigmóide).

$$\varphi(v) = \frac{e^{pv}}{e^{pv} + 1} = \frac{1}{1 + e^{-pv}}$$

$$\frac{\partial \varphi(v)}{\partial v} = pv(1-v) > 0$$



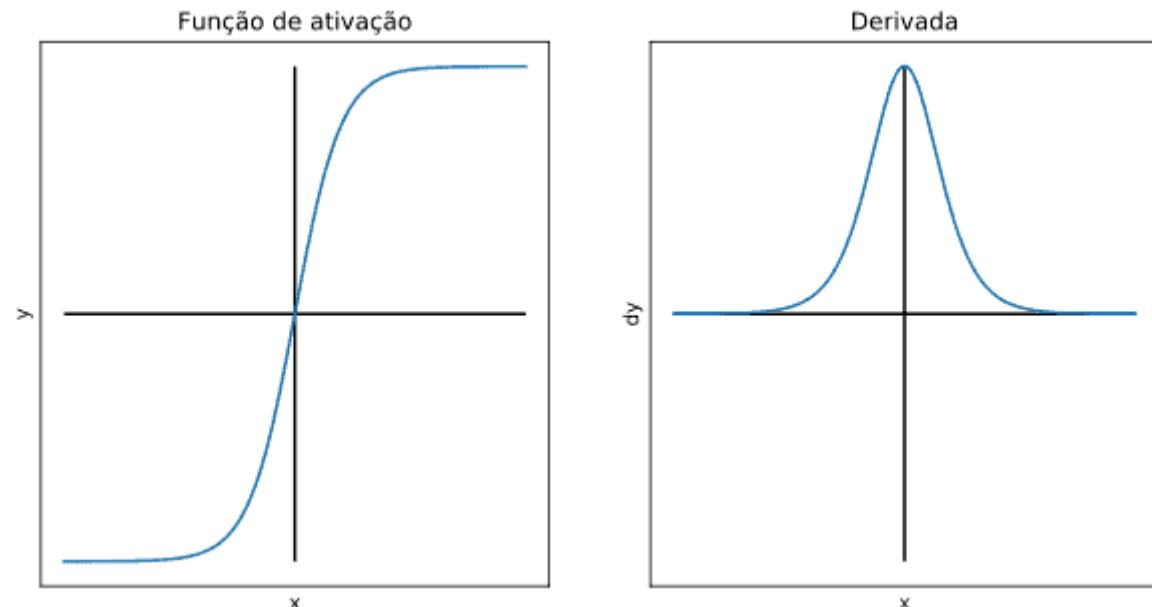
# NEURÔNIO ARTIFICIAL

## ◎ Tipos de Funções de Ativação:

### ○ Função de ativação tangente hiperbólica:

$$\varphi(v) = \tanh(pv) = \frac{e^{pv} - e^{-pv}}{e^{pv} + e^{-pv}}$$

$$\frac{\partial \varphi(v)}{\partial v} = p(1-v^2) > 0$$



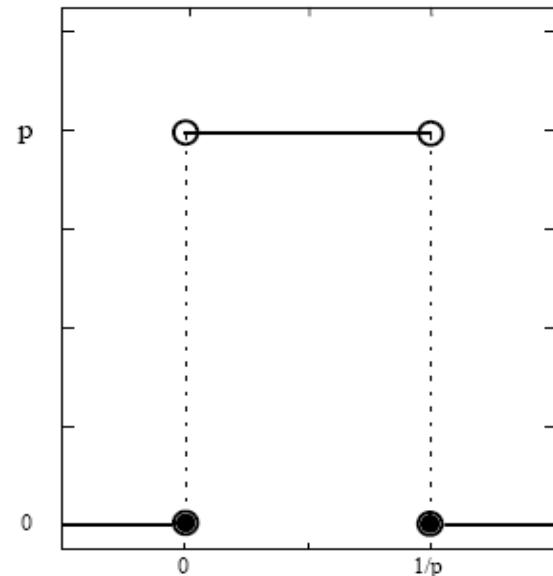
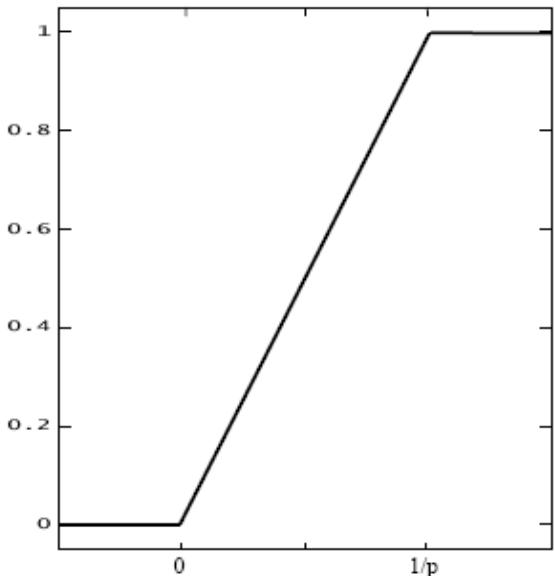
# NEURÔNIO ARTIFICIAL

## ◎ Tipos de Funções de Ativação:

### ○ Função de ativação semi-linear:

$$\varphi(v) = \begin{cases} +1 & \text{se } pv \geq 0 \\ pv & \text{se } 0 < pv < 1 \\ 0 & \text{se } pv < 0 \end{cases}$$

$$\frac{\partial \varphi(v)}{\partial v} = p$$

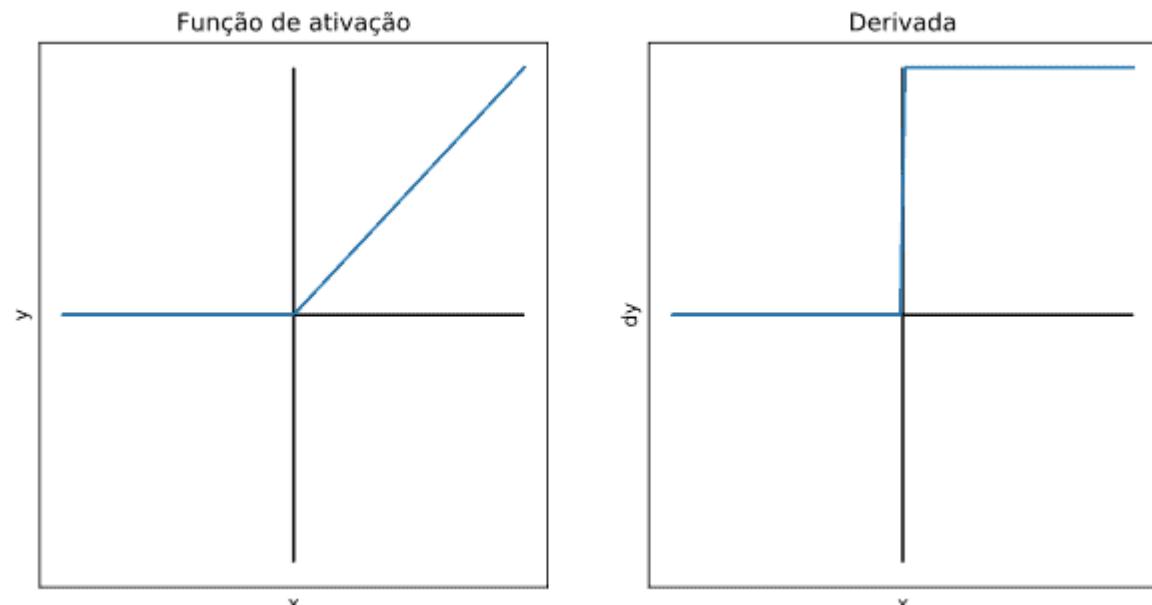


# NEURÔNIO ARTIFICIAL

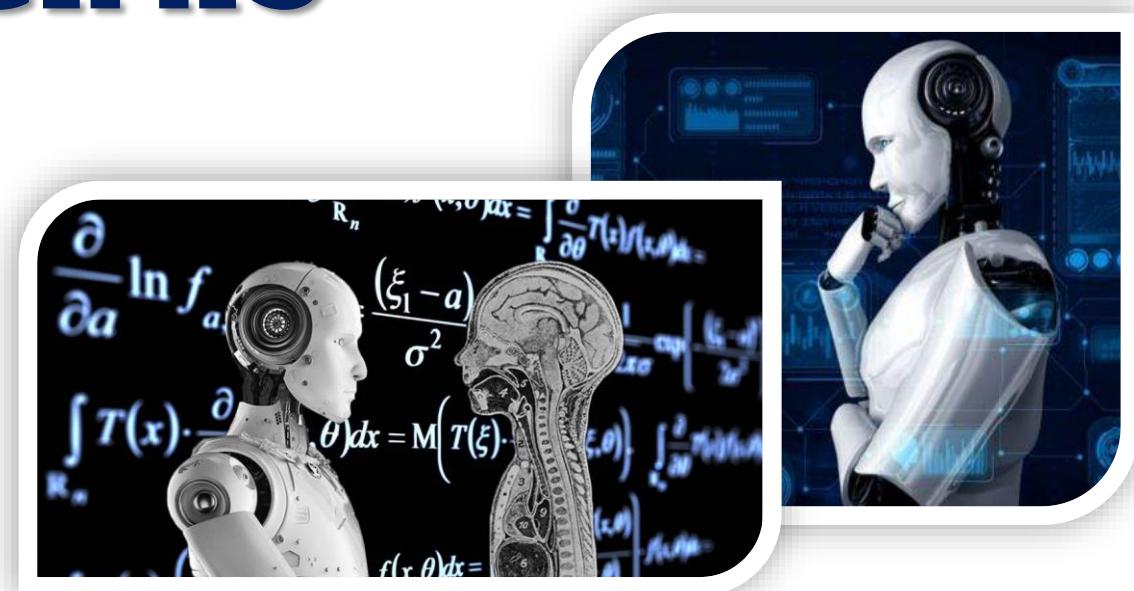
## ◎ Tipos de Funções de Ativação:

### ○ Função de ativação ReLU:

$$\phi(v) = \begin{cases} v & \text{se } v > 0 \\ 0 & \text{se } v \leq 0 \end{cases}$$



# REDES NEURAIS ARTIFICIAIS



# REDES NEURAIS ARTIFICIAIS

## ◎ PROCESSO DE APRENDIZADO:

- Um dos destaques mais relevantes das RNAs é a **capacidade de aprender a partir de amostras (exemplos)** que exprimem o comportamento do sistema;
- Isso é feito através de passos ordenados a fim de sintonizar os pesos sinápticos e limiares de seus neurônios, etapa conhecida como **processo de treinamento**;
- O **aprendizado** ocorre quando a rede neural atinge uma solução generalizada para o sistema.

# REDES NEURAIS ARTIFICIAIS

## ◎ REPRESENTAÇÃO DO CONHECIMENTO:

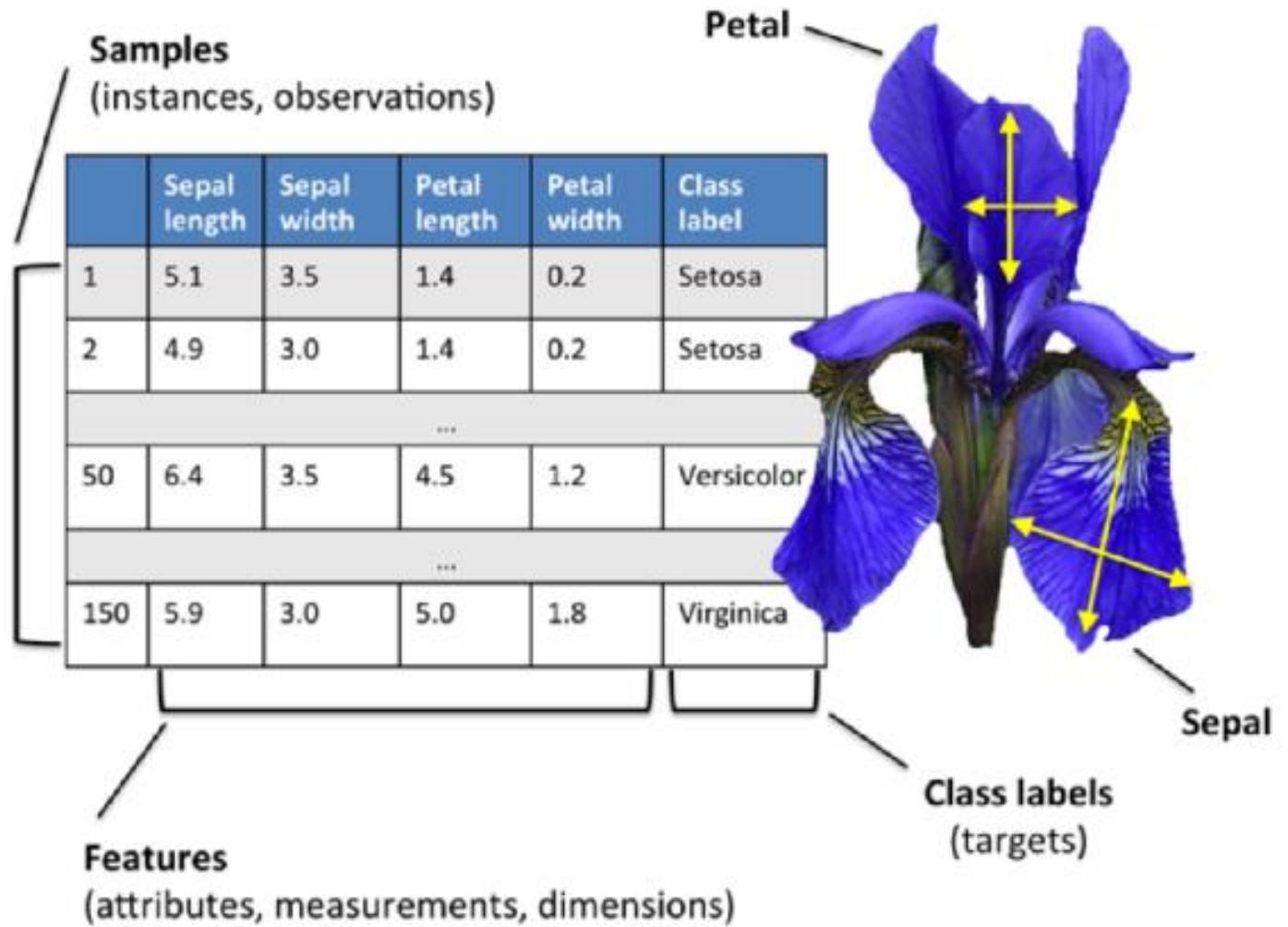
- **Tipos de exemplos:**
    - **Rotulados:** cada exemplo que representa um sinal de entrada é associado a uma resposta desejada.
    - **Não-rotulados:** ocorrências diferentes dos próprios sinais de entrada.
  - Um conjunto de pares de entrada-saída é referido como um **conjunto de dados de treinamento ou amostra de treinamento**.
  - O conjunto total de amostras disponíveis sobre o comportamento do sistema é dividido em dois subconjuntos: **subconjunto de treinamento (60 a 90% do conjunto total)** e **subconjunto de teste (10 a 40%)**;
- Cada apresentação completa dos dados de treinamento é denominado **época**.

# REDES NEURAIS ARTIFICIAIS

## ◎ REGRAS PARA O APRENDIZADO:

- Sinais de **entrada similares** provenientes de classes de eventos ou objetos similares devem produzir **representações similares** dentro da rede e devem ser classificados como pertencentes à mesma categoria;
- Itens que devem ser classificados em categorias separadas devem provocar representações bastante distintas dentro da rede;
- Se uma **característica é importante**, então deve haver um grande número de neurônios envolvidos na sua representação;
- Informações conhecidas a priori e invariância devem ser embutidas no projeto da rede.

# REDES NEURAIS ARTIFICIAIS



# REDES NEURAIS ARTIFICIAIS

## ◎ SIMILARIDADE ENTRE ENTRADAS:

$$X_i = [x_{i1}, x_{i2}, \dots, x_{iN}]^T \quad X_j = [x_{j1}, x_{j2}, \dots, x_{jN}]^T$$

### ○ Distância Euclidiana:

$$d(X_i, X_j) = \|X_i - X_j\| = \left[ \sum_{n=1}^N (x_{in} - x_{jn})^2 \right]^{1/2}$$

### ○ Produto Escalar ou Produto Interno:

$$(X_i, X_j) = X_i^T X_j = \sum_{n=1}^N x_{in} x_{jn}$$



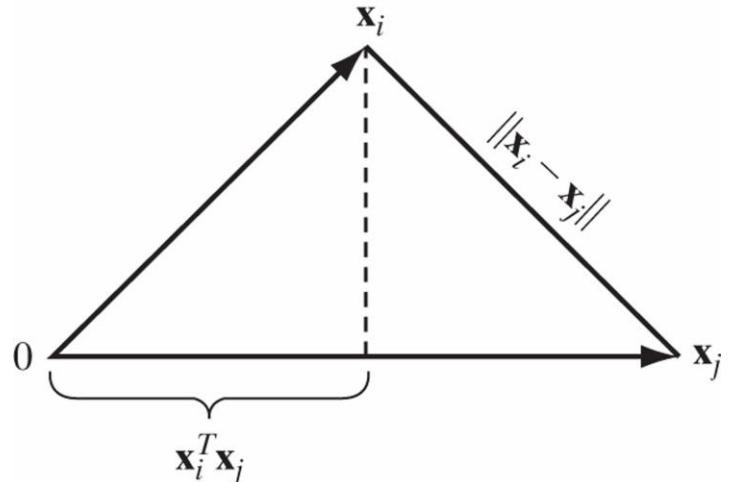
**Similaridade: inverso da distância.**

# REDES NEURAIS ARTIFICIAIS

## SIMILARIDADE ENTRE ENTRADAS:

### Relação entre as Medidas de Similaridade:

- A distância euclidiana  $\|X_i - X_j\|$  entre os vetores  $X_i$  e  $X_j$  está relacionada com a “projeção” do vetor  $X_i$  sobre o vetor  $X_j$ .
- Quanto **mais diferentes** forem  $X_i$  e  $X_j$ , **maior será a distância euclidiana**.
- Quanto **mais similares** forem  $X_i$  e  $X_j$ , **maior será o produto interno**.



# REDES NEURAIS ARTIFICIAIS

## ◎ DEFINIÇÃO DE APRENDIZAGEM:

- É o processo no qual os parâmetros livres de uma rede neural são alterados pela estimulação contínua causada pelo ambiente na qual a rede está inserida;
- O tipo do aprendizado é determinado pela maneira pela qual os parâmetros são alterados.



# REDES NEURAIS ARTIFICIAIS

## ◎ APRENDIZADO EM RNAs :

- Vários algoritmos têm sido propostos na literatura para o ajuste dos parâmetros de uma RNA.
- Por ajuste de parâmetros entende-se principalmente a **definição dos valores dos pesos** associados às conexões das RNAs.
- Esses algoritmos, referenciados como **algoritmos de treinamento**, são formados por um conjunto de regras bem definidas que especificam quando e como deve ser alterado o valor de cada peso.
- Diversos autores propuseram algoritmos de trainamento para RNAs seguindo os paradigmas de **aprendizado supervisionado, não supervisionado e por reforço**.

- **Objetivo final do aprendizado:** Obtenção de um modelo implícito dos conhecimentos adquiridos.

# REDES NEURAIS ARTIFICIAIS

## ◎ APRENDIZADO EM RNAs :

- **Algoritmos**

- Correção de erro
- Máquina de Boltzman
- Lei de Hebb
- Competição

- **Paradigmas**

- Supervisionado
- Não supervisionado
- Reforço



# REDES NEURAIS ARTIFICIAIS

## ◎ APRENDIZADO EM RNAs :

- **Correção de Erro:** Geralmente utilizados em aprendizado supervisionado, procuram ajustar os pesos da RNA de forma a reduzir os erros cometidos pela rede.
- **Hebbiano:** Frequentemente usados em aprendizado não supervisionado, são baseados na regra de Hebb, que diz que, se dois neurônios estão simultaneamente ativos, a conexão entre eles deve ser reforçada.
- **Competitivo:** Utilizados em aprendizado não supervisionado, promovem uma competição entre neurônios para definir qual ou quais devem ter seus pesos ajustados. Os neurônios que vencem a competição em geral são os que respondem mais fortemente ao objetivo apresentado aos seus terminais de entrada.
- **Termodinâmico (Boltzmann):** Algoritmos estocásticos baseados em princípios observados na metalurgia.

# REDES NEURAIS ARTIFICIAIS

## ◎ APRENDIZADO EM RNAs :

- **Aprendizado supervisionado**, quando é utilizado um agente externo que indica à rede a resposta desejada para o padrão de entrada;
- **Aprendizado não supervisionado** (auto-organização), quando não existe uma agente externo indicando a resposta desejada para os padrões de entrada;
- **Aprendizado por Reforço**, quando um crítico externo avalia a resposta fornecida pela rede.

# REDES NEURAIS ARTIFICIAIS

## APRENDIZADO SUPERVISIONADO

### Primeira Etapa



- Dispõe-se de um conjunto de dados **rotulados** (entrada  $x$ , saída  $y$ )

$$\mathcal{D} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$$

### Segunda Etapa



- O **objetivo** é construir uma função  $y = f(x)$  (**modelo**) para prever o rótulo  $y$  de uma **nova** amostra  $x$  (não-previamente observada) da mesma distribuição.

# REDES NEURAIS ARTIFICIAIS

## APRENDIZADO SUPERVISIONADO

- Dispõe-se de um conjunto de dados **rotulados** (entrada  $x$ , saída  $y$ )
- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$
- provenientes de uma distribuição  $p(x, y)$  desconhecida.
- O **objetivo** é construir uma função  $y = f(x)$  (**modelo**) para prever o rótulo  $y$  de uma **nova** amostra  $x$  (não-previamente observada) da mesma distribuição.
- **Tarefas:**
  - **Classificação:** a variável de saída é **discreta**:  $y \in \{1, \dots, K\}$ 
    - **Exemplos:** classificação de objetos em imagens, detecção de patologias, reconhecimento de fala, detecção de spam.
  - **Regressão:** a variável de saída é **contínua**:  $y \in \mathbb{R}$ 
    - **Exemplos:** predição do preço de um imóvel, predição de demanda por um serviço, avaliação de risco de um empréstimo.

# REDES NEURAIS ARTIFICIAIS

## APRENDIZADO SUPERVISIONADO

- Ambiente desconhecido pela rede neural.
- O conhecimento é apresentado por um conjunto de exemplos entrada-saída.
- A partir de um conhecimento prévio, o professor é capaz de fornecer uma resposta desejada para o vetor de treinamento;
- Os parâmetros da rede são ajustados sob a influência combinada do vetor de treinamento e do sinal de erro.
- Este ajuste é realizado passo a passo, iterativamente com o objetivo de fazer a rede emular o professor.

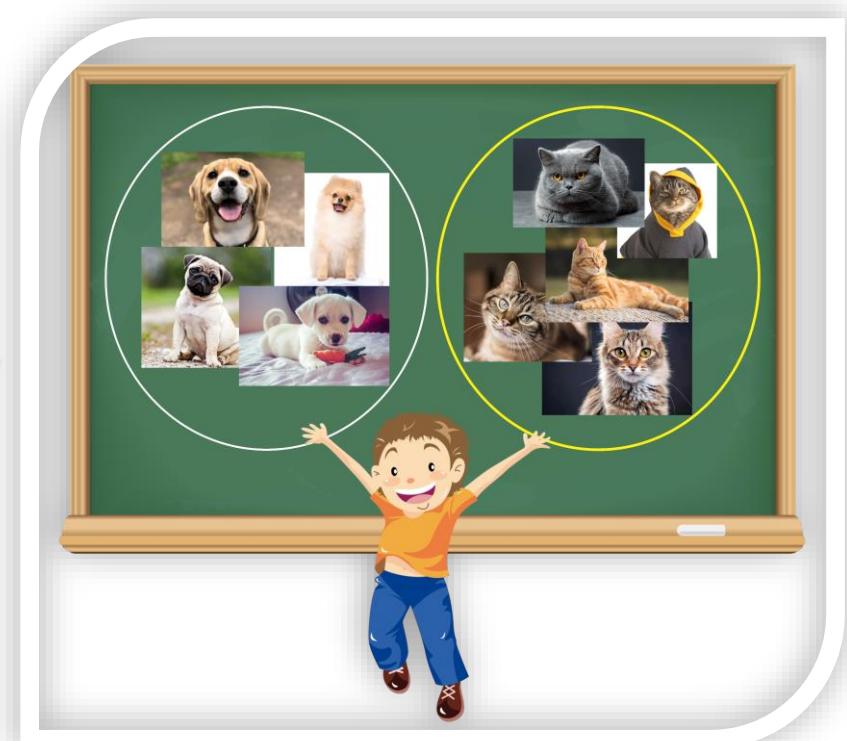
# REDES NEURAIS ARTIFICIAIS

## APRENDIZADO NÃO SUPERVISIONADO

Primeira Etapa



Segunda Etapa



# REDES NEURAIS ARTIFICIAIS

## APRENDIZADO NÃO SUPERVISIONADO

- Conjunto de dados **não rotulados**:  $\mathcal{D} = \{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$
- O **objetivo** é descobrir **propriedades** da estrutura do conjunto de dados (caracterizada pela densidade  $p(x) = p(x_1, \dots, x_n)$ ).
- **Tarefas:**
  - **Clustering:** descobrir grupos de exemplos (dados) similares.
    - *Exemplos:* segmentação de mercado, agrupamento de resultados de busca, identificação de famílias de genes, segmentação de imagens
  - **Redução de Dimensionalidade:** encontrar uma representação mais simples dos dados para agilizar algoritmos ou permitir visualização.
  - **Detecção de Anomalias:** identificar casos que fogem do padrão esperado.
    - *Exemplos:* detecção de fraudes, detecção de falhas em sistemas, monitoramento de saúde.
  - **Modelos Generativos:** gerar novos exemplos (imagens, vídeos, etc), tipicamente com características específicas.

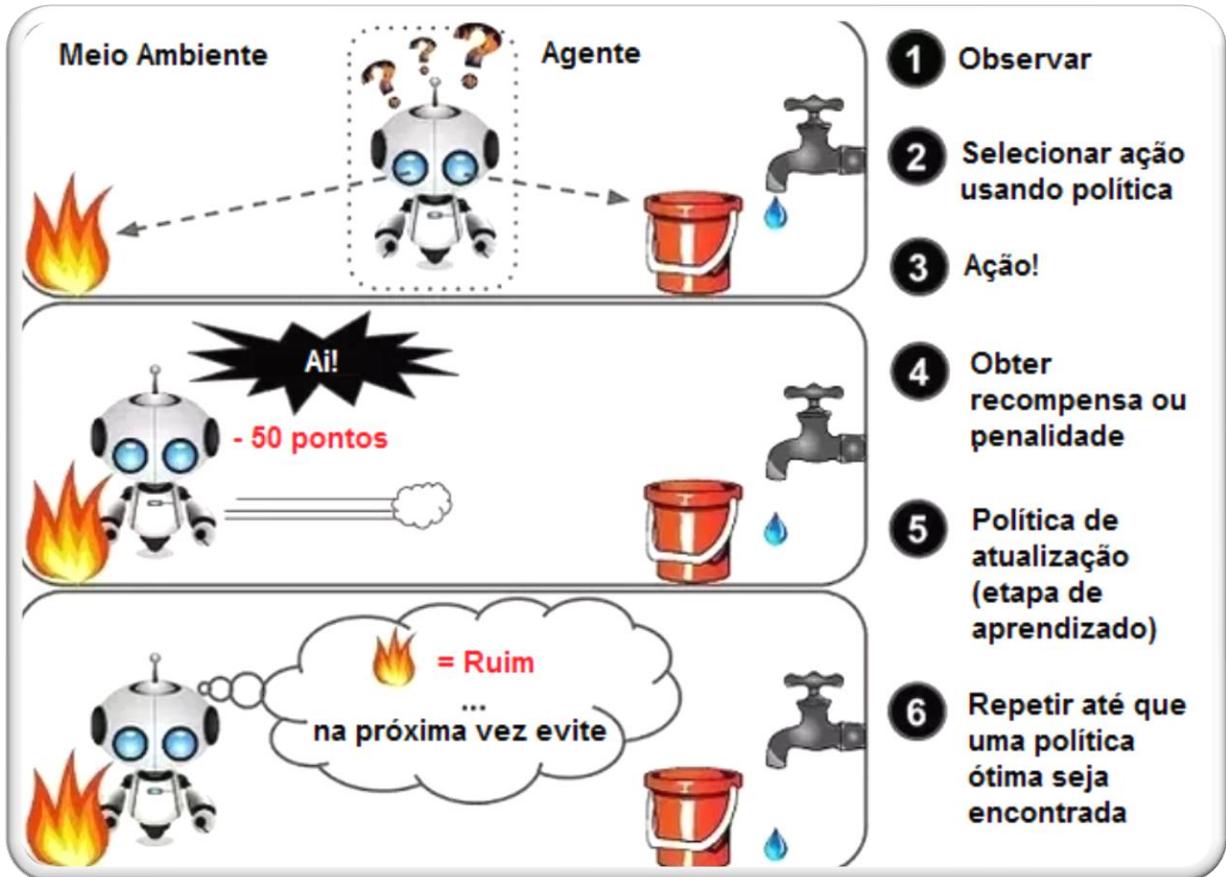
# REDES NEURAIS ARTIFICIAIS

## APRENDIZADO NÃO SUPERVISIONADO

- Não há professor para supervisionar o processo de aprendizagem.
- Não há exemplos rotulados da função a ser aprendida pela rede.
- São dadas condições para realizar um **medida independente da tarefa** da qualidade da representação que a rede deve aprender.
- Pode-se utilizar a regra de aprendizagem competitiva.

# REDES NEURAIS ARTIFICIAIS

## APRENDIZADO POR REFORÇO



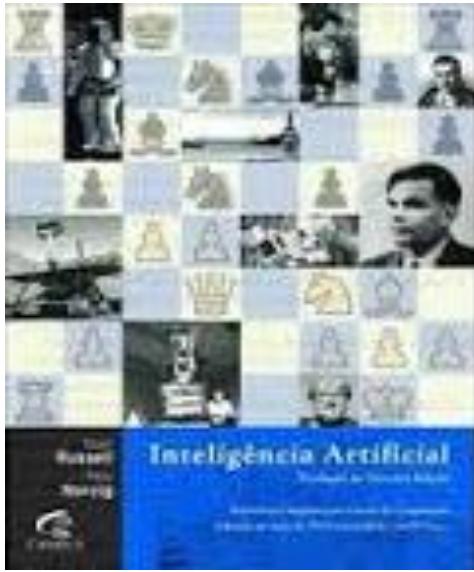
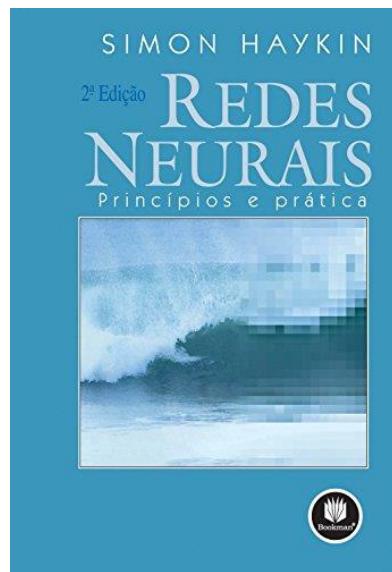
encontrar  
época  
óptima  
éles em  
uma  
série de  
atualizações  
que leva à  
política ótima

# REDES NEURAIS ARTIFICIAIS

## APRENDIZADO POR REFORÇO

- O algoritmo interage com o ambiente, recebendo um sinal de feedback (recompensa/punição) a cada tomada.
  - Formulação envolve um **conjunto de estados  $S$** , um **conjunto de ações  $A$** , uma **probabilidade de transição de estados  $p(s'|s, a)$**  e uma **recompensa associada  $\mathcal{R}_a(s, s')$** .
- O **objetivo** é descobrir e executar as melhores ações em cada situação de forma a maximizar a recompensa obtida.
- O aprendizado é feito por **tentativa e erro** e deve balancear **descoberta (exploration)** e **aproveitamento (exploitation)**.
- **Exemplos:** movimentação de robôs, jogos eletrônicos, otimização de redes de comunicação, aplicações financeiras, publicidade.
- Frequentemente combinado com técnicas de aprendizado supervisionado para aprender uma função utilidade  $Q(s, a)$ .

# REFERÊNCIAS



- HAYKIN, Simon S.; ENGEL, Paulo Martins (Paulo Martins Engel), Redes neurais: Princípios e práticas. 2 ed. São Paulo, SP: Editora Bookman, 2001, 900 p. ISBN 978-85-7307-718-6.
- RUSSELL, Stuart; NORVIG, Peter (Peter Norvig); SOUZA, Vandenberg Dantas De, Inteligência artificial. Rio de Janeiro, RJ: Editora Campus, 2004 - 2013, ISBN 978-85-352-1177-1 / 978-85-352-3701-6.
- SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade, Redes neurais artificiais: para engenharia e ciências aplicadas - curso prático. São Paulo, SP: Editora Artliber, 2010, 399 p. ISBN 978-85-88098-53-4.

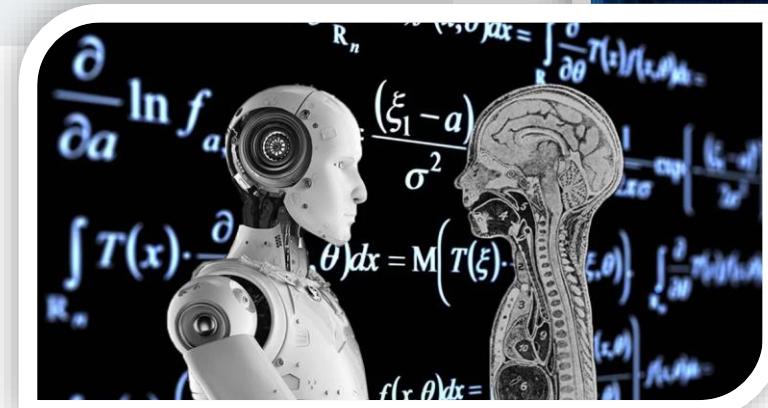
Obrigada pela Atenção!

Dúvidas?

[victoria.souto@Inatel.br](mailto:victoria.souto@Inatel.br)

# REDES PERCEPTRON

- As **Redes Neurais Artificiais (RNAs)** são baseadas em **modelos abstratos** de como pensamos que o cérebro (e os neurônios) funciona.

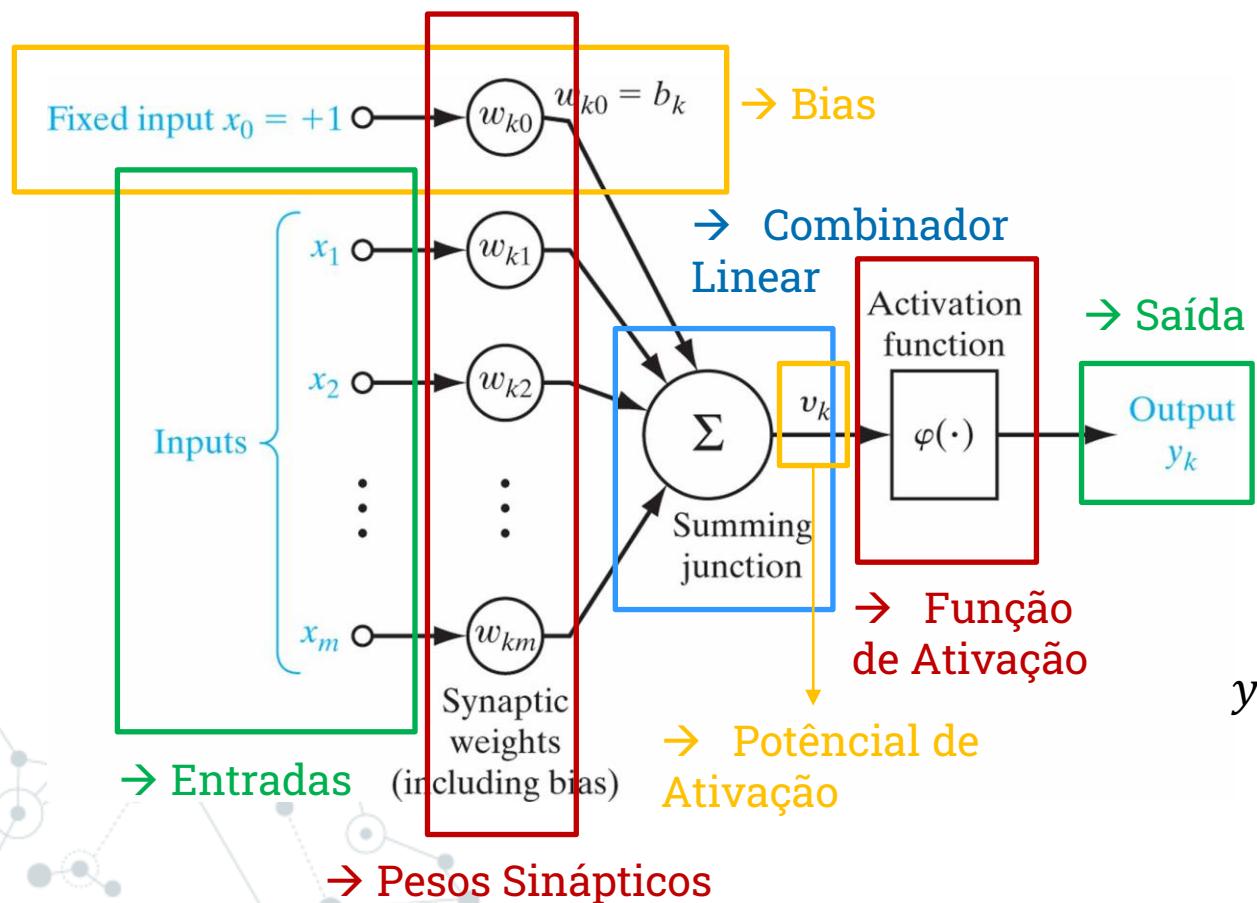


# REDES PERCEPTRON

- ◎ Primeira RNA implementada!
- ◎ Idealizado por Rosenblatt (1958), é a forma mais simples de uma rede neural artificial, pois o mesmo é constituído de **um único neurônio**;
- ◎ É considerado uma rede “*feed-forward*” (**alimentação sempre adiante, sem nenhuma realimentação de saída**);
- ◎ Sua construção é baseada no modelo de neurônio artificial de McCulloch, sendo que sua principal aplicação está na resolução de problemas envolvidos com a classificação de padrões.

# REDES PERCEPTRON

## PRINCÍPIO DE FUNCIONAMENTO.



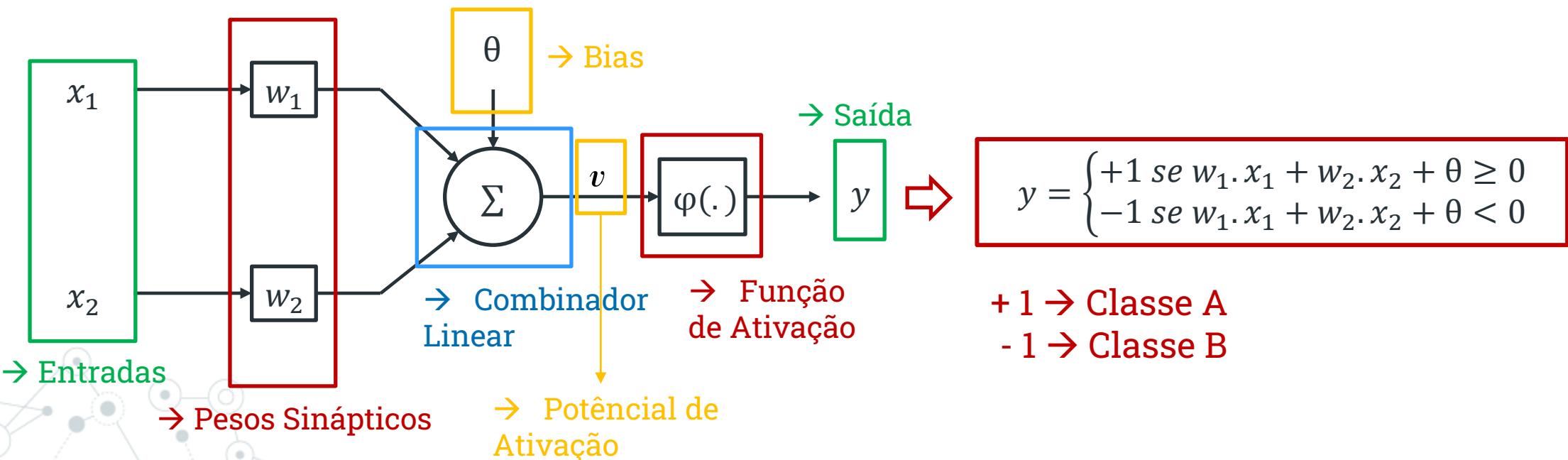
$$y_k = \begin{cases} +1 & \text{se } v_k = \sum_{i=1}^m w_i x_i + b_k = \sum_{i=0}^m w_i x_i \geq 0 \\ -1 & \text{se } v_k = \sum_{i=1}^m w_i x_i + b_k = \sum_{i=0}^m w_i x_i < 0 \end{cases}$$

$b_k = w_0 x_0$

# REDES PERCEPTRON

## ANÁLISE MATEMÁTICA DO PERCEPTRON

- Para analisar matematicamente o *Perceptron* será considerado um arquitetura com duas entradas;



# REDES PERCEPTRON

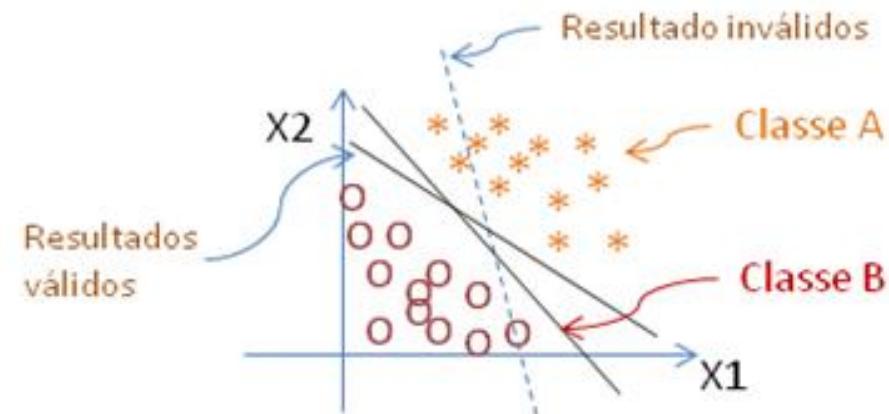
## ◎ ANÁLISE MATEMÁTICA DO PERCEPTRON

- Expressando a desigualdade através de uma equação do primeiro grau, percebe-se que a **fronteira de decisão** para este *Perceptron* de duas entradas é representada por uma reta;

$$w_1 \cdot x_1 + w_2 \cdot x_2 + \theta = 0$$

$$\rightarrow ax + by + c = 0$$

- Para a rede *Perceptron*, as classes devem ser **"linearmente separáveis"**.

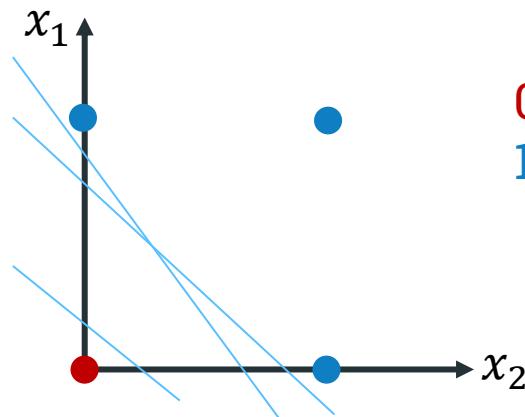


# REDES PERCEPTRON

## EXEMPLOS:

- Para o problema do **OU** lógico, pode-se utilizar um *Perceptron* de camada única?

$x_1$	$x_2$	$x_1 \text{ OU } x_2$
0	0	0
0	1	1
1	0	1
1	1	1



$0 \rightarrow \text{Classe A}$   
 $1 \rightarrow \text{Classe B}$

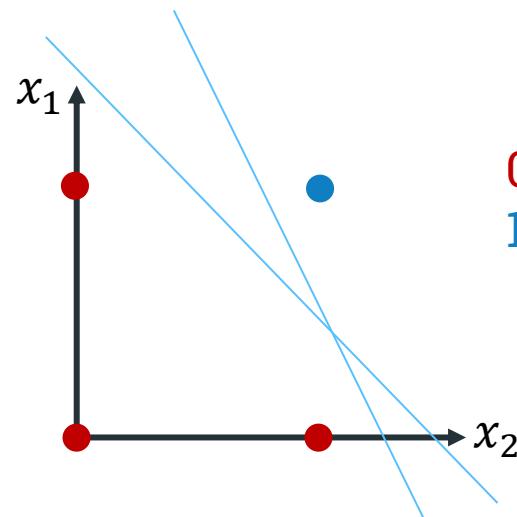
Linearmente Separável!

# REDES PERCEPTRON

## EXEMPLOS:

- Para o problema do **E** lógico, pode-se utilizar um *Perceptron* de camada única?

$x_1$	$x_2$	$x_1 \text{ E } x_2$
0	0	0
0	1	0
1	0	0
1	1	1



0 → Classe A  
1 → Classe B

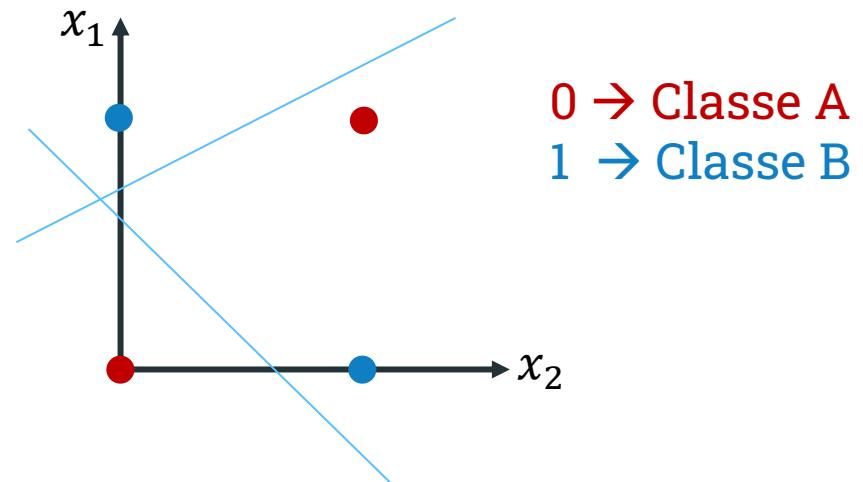
Linearmente Separável!

# REDES PERCEPTRON

## EXEMPLOS:

- Para o problema do **OU EXCLUSIVO** lógico, pode-se utilizar um *Perceptron* de camada única?

$x_1$	$x_2$	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0



Não é Linearmente Separável!

# REDES PERCEPTRON

## ◎ TREINAMENTO PERCEPTRON:

- O processo de treinamento do *Perceptron* está associado ao ajuste dos pesos sinápticos e do limiar da rede com o objetivo de classificar padrões;
- Para o *Perceptron*, a regra de aprendizado utilizada é a regra de Hebb;
- **Resumidamente:**
  - Se a saída reproduzida é **coincidente** com a saída desejada, os pesos sinápticos e limiar da rede serão **mantidos**;
  - Caso contrário, os pesos sinápticos e limiar serão **ajustados** (incrementados/decrementados) proporcionalmente aos valores de seus sinais de entrada → Este processo é **repetido sequencialmente** para todas as amostras de treinamento até que a **saída do Perceptron seja similar a saída desejada para cada amostra**;

# REDES PERCEPTRON

## ◎ TREINAMENTO PERCEPTRON:

- A Rede Perceptron é treinada por um **Algoritmo Supervisionado de Correção de Erro** e usa a função de ativação do tipo limiar. Durante o seu treinamento, para um objeto  $x_i$ , os pesos são ajustados de acordo com:

$$\begin{cases} w_i^{Atual} = w_i^{Anterior} + \eta \cdot (d^{(k)} - y) \cdot x^{(k)} \\ \theta_i^{Atual} = \theta_i^{Anterior} + \eta \cdot (d^{(k)} - y) \cdot x^{(k)} \end{cases}$$

- Onde:

- $w_i^{Atual}$  → Pesos Novos
- $\theta_i^{Atual}$  → Bias Novos
- $\eta$  → Taxa de Aprendizado (0,1]
- $x^{(k)}$  → Entrada da  $k$ -ésima amostra
- $d^{(k)}$  → Saída Desejada para  $k$ -ésima amostra
- $y$  → Saída da Rede para a  $k$ -ésima amostra
- $w_i^{Anterior}$  → Pesos Atuais
- $\theta_i^{Anterior}$  → Bias Atuais

# REDES PERCEPTRON

## ◎ TREINAMENTO PERCEPTRON:

- Em termos de implementação computacional, fica mais fácil tratar as expressões na **forma vetorial**:

$$w_i^{Atual} = w_i^{Anterior} + \eta \cdot (d^{(k)} - y) \cdot x^{(k)}$$
$$\vec{w}^{Atual} = \vec{w}^{Anterior} + \Delta v$$

- Onde:

- $w = [w_0 \ w_1 \ w_2 \ ... \ w_n]$  → Vetor contendo o bias e os pesos.
- $x^{(k)} = [\theta, \ x_1^{(k)}, \ x_2^{(k)} \ ... \ x_n^{(k)}]$  → Vetor das amostras de treinamento.
- $d^{(k)}$  → Valor desejado para a  $k$ -ésima amostra de treinamento;
- $y$  → Valor de saída produzido pelo *Perceptron*;
- $\eta$  → Constante que define a taxa de aprendizagem
  - *Normalmente adota-se  $0 < \eta < 1$ . Quando muito grande, não converge. Se muito pequena, não chega no resultado.*

# TREINAMENTO PERCEPTRON

- 1) Obter conjunto de amostras de treinamento  $\{x^{(k)}\}$ ;
- 2) Associar a saída desejada  $\{d^{(k)}\}$  para cada amostra obtida;
- 3) Iniciar o vetor  $w$  com valores aleatórios pequenos  $[-1, 1]$  ou  $[0, 1]$ ;
- 4) Especificar a taxa de aprendizagem  $\{\eta\} \rightarrow (0,1]$ ;
- 5) Iniciar o contador de número de épocas  $\{\text{épocas} \leftarrow 0\}$ ;
- 6) Repetir as instruções:

6.1) erro  $\leftarrow$  "inexiste";  $\longrightarrow$  Critério de Parada

6.2) Para todos pares de treinamento  $\{x^{(k)}, d^{(k)}\}$ , faça:

6.2.1)  $v \leftarrow w^T * x^k$ ;  $\longrightarrow$  Potencial de Ativação

6.2.2)  $y \leftarrow \text{degrau}(v)$ ; (sign no Matlab)  $\longrightarrow$  Função de Ativação

6.2.3) Se  $y \neq d^{(k)}$   $\longrightarrow$  ERRO

6.2.3.1) então  $\begin{cases} w \leftarrow w + \eta * (d^{(k)} - y) * x^{(k)} \\ \text{erro} \leftarrow \text{"existe"} \end{cases}$   $\longrightarrow$  Atualiza os Pesos

6.3)  $\text{época} \leftarrow \text{época} + 1$ ;  $\longrightarrow$  Incrementa o Número de Épocas

Até que:  $\text{erro} == \text{"inexiste"}$   $\longrightarrow$   **$w$  Definidos  $\rightarrow$  Treinamento Realizado!**



Aprendizado Supervisionado

# FUNCIONAMENTO PERCEPTRON

- 1) Obter a amostra a ser classificada  $\{x\}$ ;
- 2) Utilizar o vetor  $w$  ajustado durante o treinamento;

## 3) Executar as seguintes instruções:

3.1)  $v \leftarrow w^T * x$ ;  $\longrightarrow$  Potencial de Ativação

3.2)  $y \leftarrow \text{degrau}(v)$ ; (sign no Matlab)  $\longrightarrow$  Função de Ativação

3.3) Se  $y == -1$

3.3.1) Então: amostra  $x \in \{\text{Classe } A\}$

3.4) Se  $y == 1$

3.4.1) Então: amostra  $x \in \{\text{Classe } B\}$

CLASSIFICAÇÃO

# REDES PERCEPTRON

## EXEMPLO TREINAMENTO PERCEPTRON:

- Supondo um problema a ser mapeado pelo *Perceptron* com **duas entradas**  $\{x_1, x_2\}$ ;
- Para um conjunto de quatro **amostras de treinamento** constituídas dos seguintes valores:  
 $\Omega^{(x)} = \{[2.0 \ 3.5]; [6.8 \ 5.3]; [2.0 \ 2.5]; [8.1 \ 4.2]\}$ .
- Considerando-se ainda que os respectivos **valores de saída** para cada uma das amostras seja dado por  $\Omega^{(d)} = \{[-1]; [+1]; [-1]; [+1]\}$ .
- Escolhendo aleatoriamente os **pesos sinápticos** iniciais:  $w = \{0.84; 0.68; 0.88\}$ .

**Características**

$$\Omega^{(x)} = \begin{bmatrix} x_0 & x_1 & x_2 \\ -1 & 2.0 & 3.5 \\ -1 & 6.8 & 5.3 \\ -1 & 2.0 & 2.5 \\ -1 & 8.1 & 4.2 \end{bmatrix}$$

**Bias**

**Saídas Esperadas**

$$\Omega^{(d)} = \begin{bmatrix} d^{(1)} \\ d^{(2)} \\ d^{(3)} \\ d^{(4)} \end{bmatrix} = \begin{bmatrix} -1 \\ +1 \\ -1 \\ +1 \end{bmatrix}$$

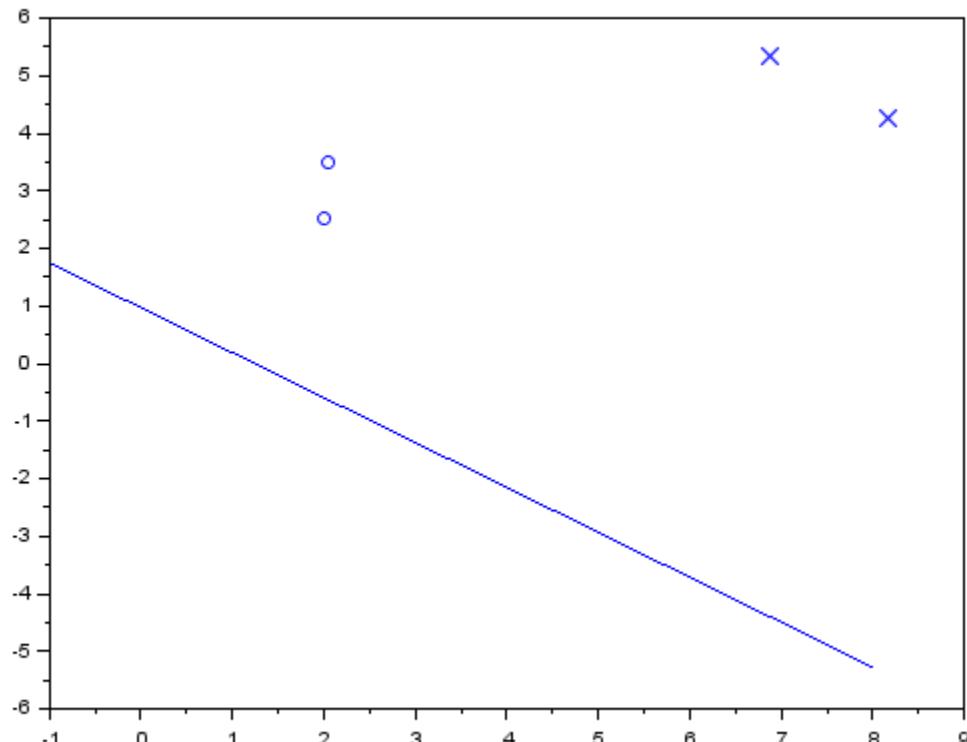
-1 → Classe A - o  
+1 → Classe B - x

# REDES PERCEPTRON

## Exemplo Treinamento Perceptron:

### Uma Época de Treinamento

$$w = \{0.84; 0.68; 0.88\}.$$

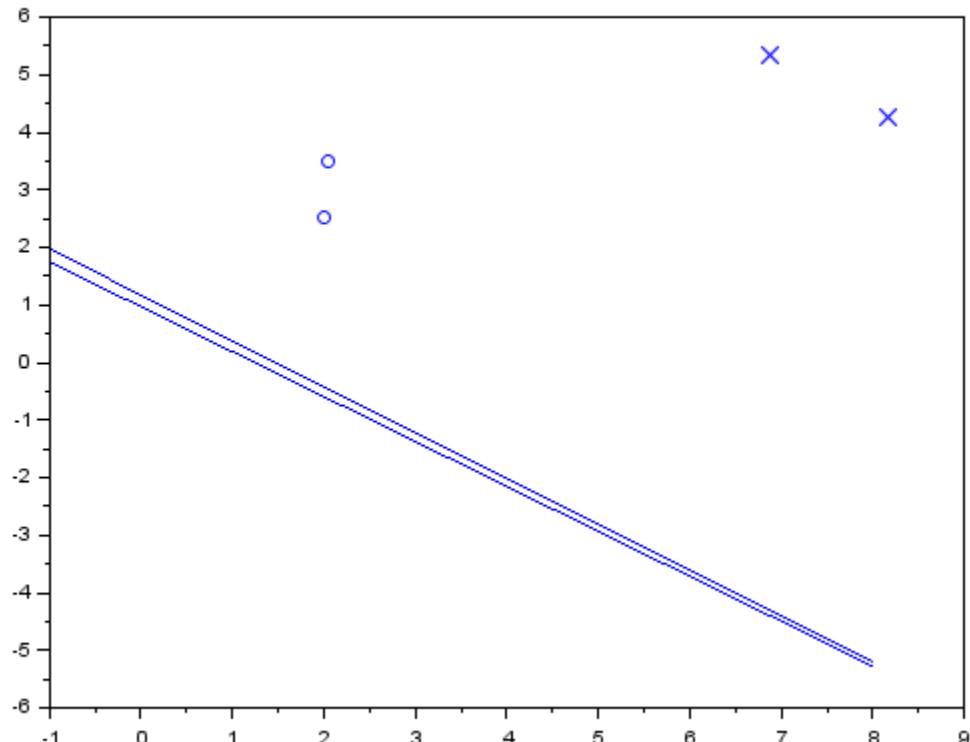


# REDES PERCEPTRON

## EXEMPLO TREINAMENTO PERCEPTRON:

- Duas Épocas de Treinamento

$$w = \{0.93; 0.52; 0.64\}.$$

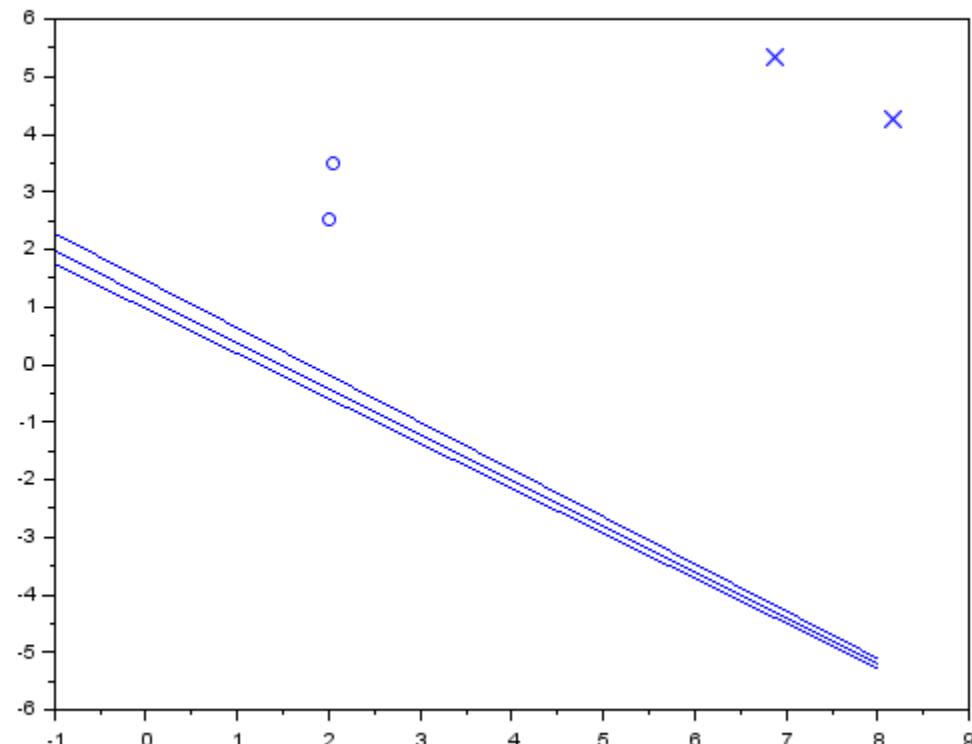


# REDES PERCEPTRON

## Exemplo Treinamento Perceptron:

- Três Épocas de Treinamento

$$w = \{0.97; 0.44; 0.51\}.$$

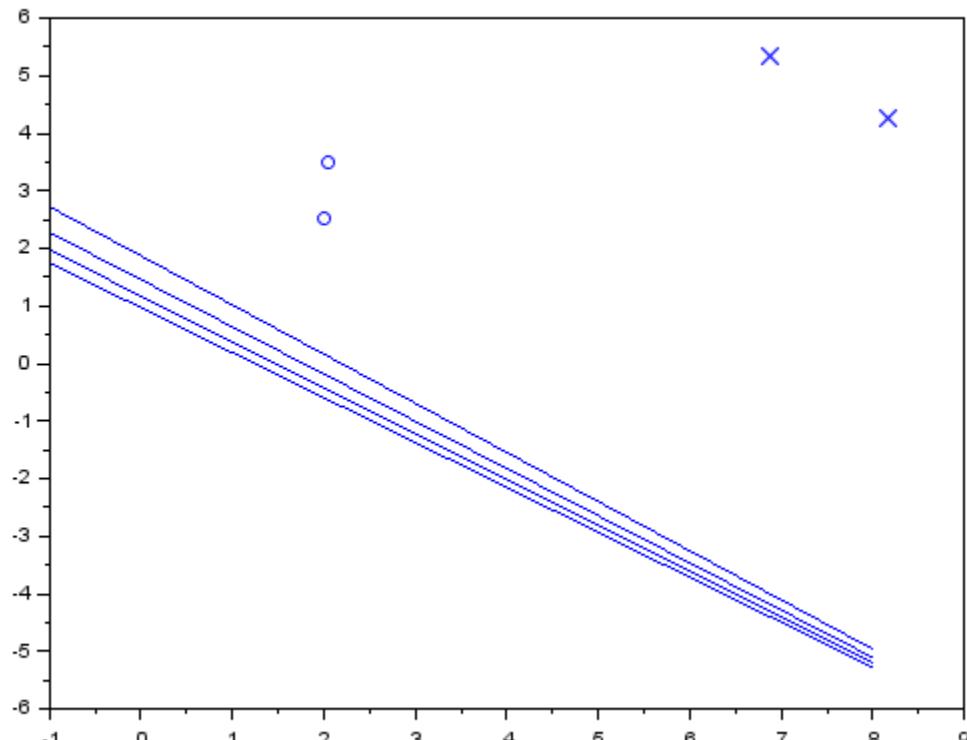


# REDES PERCEPTRON

## Exemplo Treinamento Perceptron:

### Quatro Épocas de Treinamento

$$w = \{1.0; 0.36; 0.39\}.$$

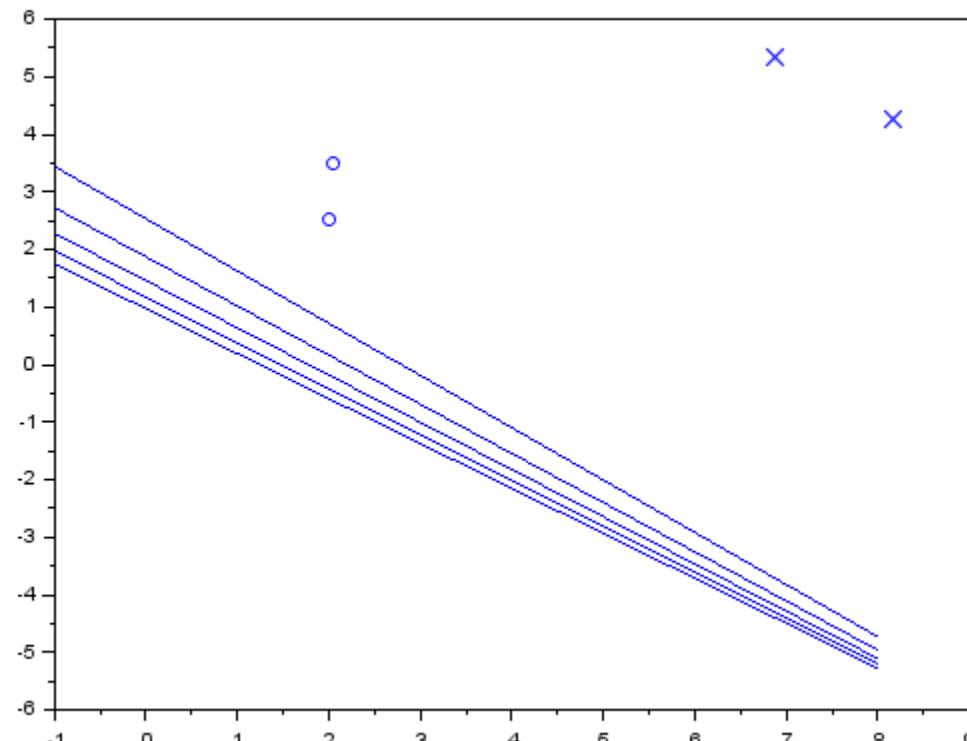


# REDES PERCEPTRON

## Exemplo Treinamento Perceptron:

### Cinco Épocas de Treinamento

$$w = \{1.04; 0.28; 0.27\}.$$

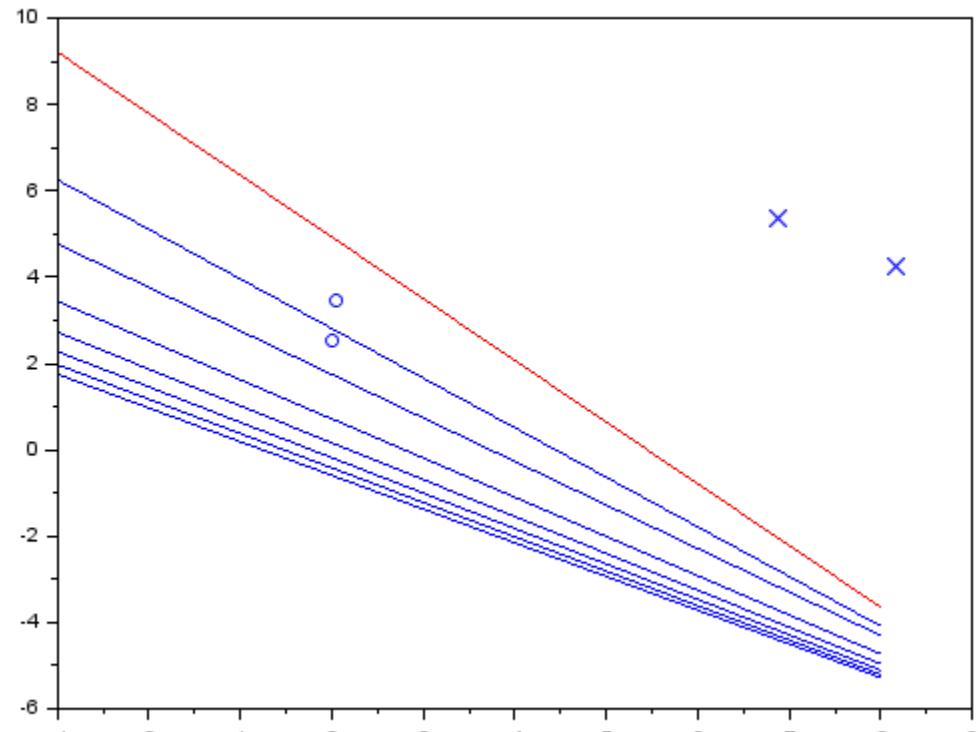


# REDES PERCEPTRON

## Exemplo Treinamento Perceptron:

Após Oito Épocas de Treinamento

$$w = \{1.09; 0.2; 0.14\}.$$

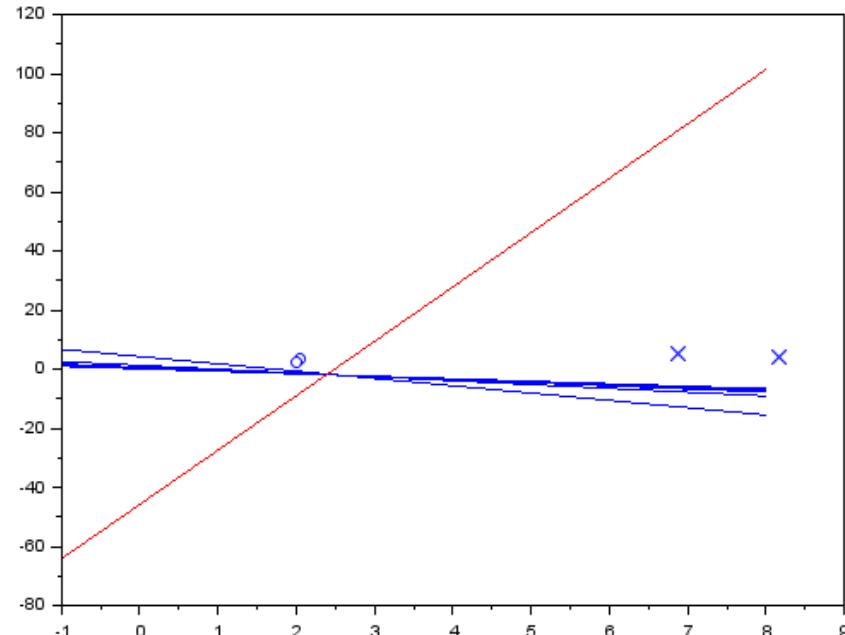


# REDES PERCEPTRON

## Exemplo Treinamento Perceptron:

- Usando o mesmo problema anterior porém escolhendo outros valores, aleatoriamente, para os pesos sinápticos iniciais, encontra-se uma solução após **7 épocas de treinamento** resultando nos pesos sinápticos finais

$$w = \{0.28; 0.11; -0.01\}.$$



# REDES PERCEPTRON

## EXERCÍCIO:

- Porta Lógica OU → Considere os dados a seguir:

$$\Omega^{(x)} = \begin{bmatrix} -1 & -1 \\ -1 & +1 \\ +1 & -1 \\ +1 & +1 \end{bmatrix}$$

Entradas de Treinamento

$$\Omega^{(d)} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

Saídas Desejadas

$$\theta = -1$$
$$w_\theta = 0,1$$

Baias

$$w = [0,5 \quad 0,7]$$

Pesos Sinapticos

$$\eta = 0,1$$

Taxa de Aprendizagem

# REDES PERCEPTRON

## EXERCÍCIO:

- Porta Lógica OU → Considere os dados a seguir:

$$\Omega^{(x)} = \begin{bmatrix} -1 & -1 \\ -1 & +1 \\ +1 & -1 \\ +1 & +1 \end{bmatrix} \quad w = [0,5 \quad 0,7]$$
$$\theta = -1$$
$$w_\theta = 0,1$$

- Adicionando  $\theta$  à  $\Omega^{(x)}$  e  $w_\theta$  à  $w$ , tem-se:

$$\Omega^{(x)} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & -1 \\ -1 & +1 & +1 \end{bmatrix} \quad w = [0,1 \quad 0,5 \quad 0,7]$$

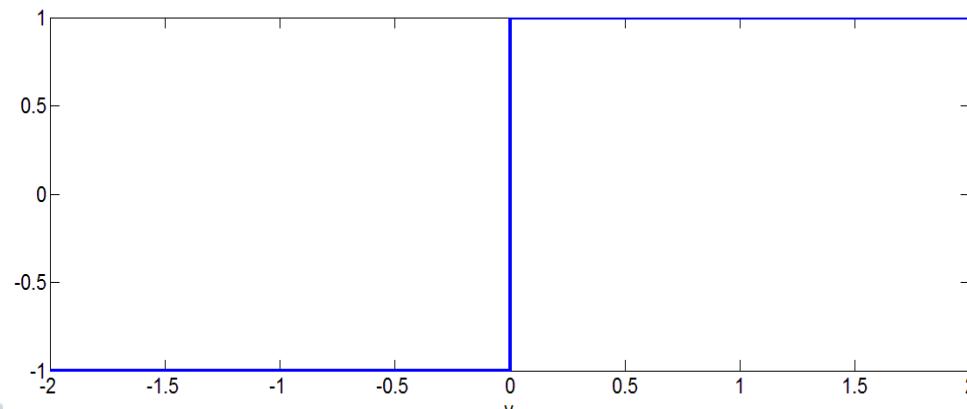
# REDES PERCEPTRON

## EXERCÍCIO:

- Porta Lógica OU → Considere os dados a seguir:

$$\Omega^{(d)} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

- Observando os valores que  $\Omega^{(d)}$  assume, pode-se adotar a função de ativação Heaviside Simétrica (sinal).



$$\varphi(v) = \begin{cases} +1 & \text{se } v > 0 \\ 0 & \text{se } v = 0 \\ -1 & \text{se } v < 0 \end{cases}$$

# REDES PERCEPTRON

- ◎ **EXERCÍCIO:**
  - **Porta Lógica OU.**
  - Determine o Número de Épocas de Treinamento e os Pesos Sinápticos Resultantes.

# REDES PERCEPTRON

## EXERCÍCIO:

### Porta Lógica OU.

$$\Omega^{(x)} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & -1 \\ -1 & +1 & +1 \end{bmatrix}$$

Entradas de Treinamento

$$w = [0,1 \quad 0,5 \quad 0,7]$$

Pesos Sinapticos

$$\Omega^{(d)} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

Saídas Desejadas

$$\eta = 0,1$$

Taxa de Aprendizagem

$$\varphi(v) = \begin{cases} +1 & \text{se } v > 0 \\ 0 & \text{se } v = 0 \\ -1 & \text{se } v < 0 \end{cases}$$

→ Função de Ativação

### Primeira Época

1. Calcular o Potencial de Ativação e Saída

$$v^{(k)} \leftarrow w * \Omega^{(k)}$$

$$v^{(0)} = (0,1)(-1) + (0,5)(-1) + (0,7)(-1) = -1,3$$
$$y^{(0)} = \text{sign}(-1,3) = -1$$

$$v^{(1)} = (0,1)(-1) + (0,5)(-1) + (0,7)(+1) = 0,1$$
$$y^{(1)} = \text{sign}(0,1) = 1$$

$$v^{(2)} = (0,1)(-1) + (0,5)(+1) + (0,7)(-1) = -0,3$$
$$y^{(2)} = \text{sign}(-0,3) = -1 \rightarrow \text{ERRO}$$

2. Atualiza os Pesos

$$w = w^{\text{Anterior}} + \eta \cdot (\Omega^{(d)} - y) \cdot \Omega^{(k)}$$

$$w = [0,1 \ 0,5 \ 0,7] + (0,1)(1 - (-1))[-1 \ +1 \ -1]$$

$$w = [-0,1 \ 0,7 \ 0,5]$$

# REDES PERCEPTRON

## EXERCÍCIO:

Porta Lógica OU.

$$\Omega^{(x)} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & -1 \\ -1 & +1 & +1 \end{bmatrix}$$

Entradas de Treinamento

$$w = [-0,1 \quad 0,7 \quad 0,5]$$

Pesos Sinapticos

$$\Omega^{(d)} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

Saídas Desejadas

$$\eta = 0,1$$

Taxa de Aprendizagem

$$\varphi(v) = \begin{cases} +1 & \text{se } v > 0 \\ 0 & \text{se } v = 0 \\ -1 & \text{se } v < 0 \end{cases} \rightarrow \text{Função de Ativação}$$

Nesse ponto, o treinamento já pode encerrar?

*NÃO, pois houve erro de classificação em pelo menos uma amostra do conjunto.*

*Portanto, deve-se realizar mais uma época de treinamento.*

# REDES PERCEPTRON

## EXERCÍCIO:

### Porta Lógica OU.

$$\Omega^{(x)} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & -1 \\ -1 & +1 & +1 \end{bmatrix}$$

Entradas de Treinamento

$$w = [-0,1 \quad 0,7 \quad 0,5]$$

Pesos Sinápticos

$$\Omega^{(d)} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

Saídas Desejadas

$$\eta = 0,1$$

Taxa de Aprendizagem

$$\varphi(v) = \begin{cases} +1 & se v > 0 \\ 0 & se v = 0 \\ -1 & se v < 0 \end{cases}$$

→ Função de Ativação

### Segunda Época

#### 1. Calcular o Potencial de Ativação e Saída

$$v^{(k)} \leftarrow w * \Omega^{(k)}$$

$$v^{(0)} = (-0,1)(-1) + (0,7)(-1) + (0,5)(-1) = -1,1$$
$$y^{(0)} = \text{sign}(-1,1) = -1$$

$$v^{(1)} = (-0,1)(-1) + (0,7)(-1) + (0,5)(+1) = -0,1$$
$$y^{(1)} = \text{sign}(-0,1) = -1 \rightarrow \text{ERRO}$$

#### 2. Atualiza os Pesos

$$w = w^{\text{Anterior}} + \eta \cdot (\Omega^{(d)} - y) \cdot \Omega^{(k)}$$

$$w = [-0,1 \ 0,7 \ 0,5] + (0,1)(1 - (-1))[-1 \ -1 \ +1]$$

$$w = [-0,3 \ 0,5 \ 0,7]$$

# REDES PERCEPTRON

## EXERCÍCIO:

Porta Lógica OU.

$$\Omega^{(x)} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & -1 \\ -1 & +1 & +1 \end{bmatrix}$$

Entradas de Treinamento

$$w = [-0,3 \quad 0,5 \quad 0,7]$$

Pesos Sinapticos

$$\Omega^{(d)} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

Saídas Desejadas

$$\eta = 0,1$$

Taxa de Aprendizagem

$$\varphi(v) = \begin{cases} +1 & \text{se } v > 0 \\ 0 & \text{se } v = 0 \\ -1 & \text{se } v < 0 \end{cases} \rightarrow \text{Função de Ativação}$$

Nesse ponto, o treinamento já pode encerrar?

*NÃO, pois houve erro de classificação em pelo menos uma amostra do conjunto.*

*Portanto, deve-se realizar mais uma época de treinamento.*

# REDES PERCEPTRON

## EXERCÍCIO:

### Porta Lógica OU.

$$\Omega^{(x)} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & -1 \\ -1 & +1 & +1 \end{bmatrix}$$

Entradas de Treinamento

$$w = [-0,3 \quad 0,5 \quad 0,7]$$

Pesos Sinapticos

$$\Omega^{(d)} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

Saídas Desejadas

$$\eta = 0,1$$

Taxa de Aprendizagem

$$\varphi(v) = \begin{cases} +1 & se v > 0 \\ 0 & se v = 0 \\ -1 & se v < 0 \end{cases}$$

→ Função de Ativação

### Segunda Época

1. Calcular o Potencial de Ativação e Saída

$$v^{(k)} \leftarrow w * \Omega^{(k)}$$

$$v^{(0)} = (-0,3)(-1) + (0,5)(-1) + (0,7)(-1) = -0,9$$
$$y^{(0)} = \text{sign}(-0,9) = -1$$

$$v^{(1)} = (-0,3)(-1) + (0,5)(-1) + (0,7)(+1) = 0,5$$
$$y^{(1)} = \text{sign}(0,5) = 1$$

$$v^{(2)} = (-0,3)(-1) + (0,5)(1) + (0,7)(-1) = 0,1$$
$$y^{(2)} = \text{sign}(0,1) = 1$$

$$v^{(3)} = (-0,3)(-1) + (0,5)(1) + (0,7)(1) = 1,5$$
$$y^{(3)} = \text{sign}(1,5) = 1$$

# REDES PERCEPTRON

## EXERCÍCIO:

Porta Lógica OU.

$$\Omega^{(x)} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & +1 \\ -1 & +1 & -1 \\ -1 & +1 & +1 \end{bmatrix}$$

Entradas de Treinamento

$$w = [-0,3 \quad 0,5 \quad 0,7]$$

Pesos Sinápticos

$$\varphi(v) = \begin{cases} +1 & \text{se } v > 0 \\ 0 & \text{se } v = 0 \\ -1 & \text{se } v < 0 \end{cases}$$

$$\Omega^{(d)} = \begin{bmatrix} -1 \\ +1 \\ +1 \\ +1 \end{bmatrix}$$

Saídas Desejadas

$$\eta = 0,1$$

Taxa de Aprendizagem

→ Função de Ativação

Nesse ponto, o treinamento já pode encerrar?

*SIM, pois não houve erro de classificação nas amostras do conjunto.*

O resultado do treinamento é, portanto, o vetor de pesos sinápticos ajustados.

$$w = [-0,3 \quad 0,5 \quad 0,7]$$

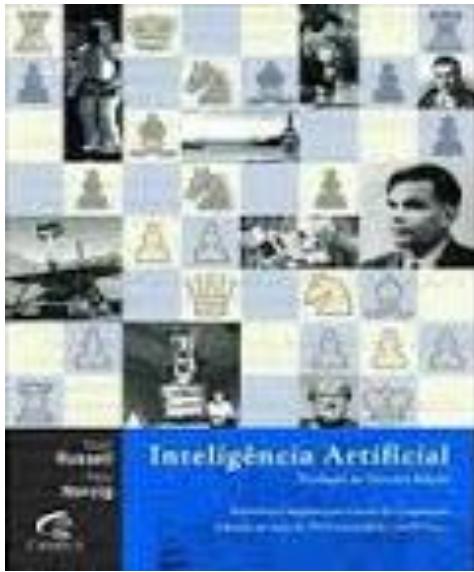
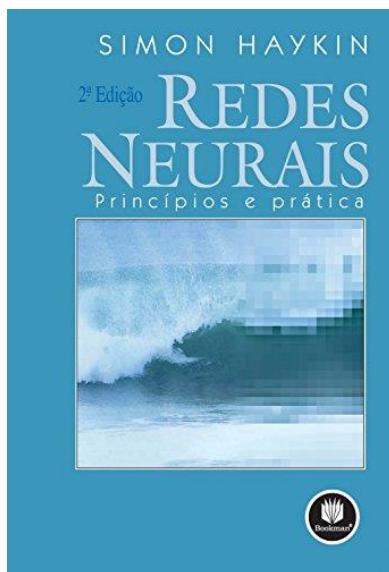
Resultado do Treinamento

# REDES PERCEPTRON

## ◎ OBSERVAÇÕES:

- Podem existir **infinitas soluções** para a rede dependendo dos pesos sinápticos iniciais escolhidos. Logo a **solução não é ótima e o número de épocas varia**;
- A rede divergirá se o problema não for linearmente separável;
- Usando faixas de separabilidade entre as classes forem muito estreitas, o processo de treinamento pode implicar em **instabilidade**. Neste caso utiliza-se uma **taxa de aprendizagem  $\{\eta\}$  bem pequena**.
- Quanto mais próxima a superfície de decisão estiver da fronteira de separabilidade, menos épocas serão necessárias para a rede convergir.
- A normalização das entradas para domínios apropriados contribui para o incremento do desempenho da rede.

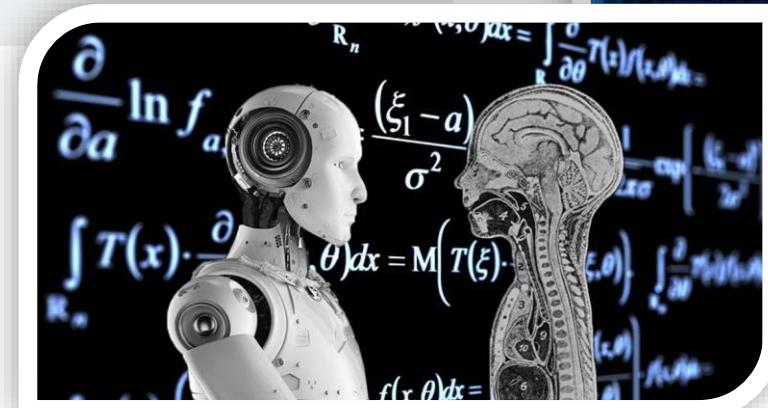
# REFERÊNCIAS



- HAYKIN, Simon S.; ENGEL, Paulo Martins (Paulo Martins Engel), Redes neurais: Princípios e práticas. 2 ed. São Paulo, SP: Editora Bookman, 2001, 900 p. ISBN 978-85-7307-718-6.
- RUSSELL, Stuart; NORVIG, Peter (Peter Norvig); SOUZA, Vandenberg Dantas De, Inteligência artificial. Rio de Janeiro, RJ: Editora Campus, 2004 - 2013, ISBN 978-85-352-1177-1 / 978-85-352-3701-6.
- SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade, Redes neurais artificiais: para engenharia e ciências aplicadas - curso prático. São Paulo, SP: Editora Artliber, 2010, 399 p. ISBN 978-85-88098-53-4.

# REDES ADALINE

- As **Redes Neurais Artificiais (RNAs)** são baseadas em **modelos abstratos** de como pensamos que o cérebro (e os neurônios) funciona.

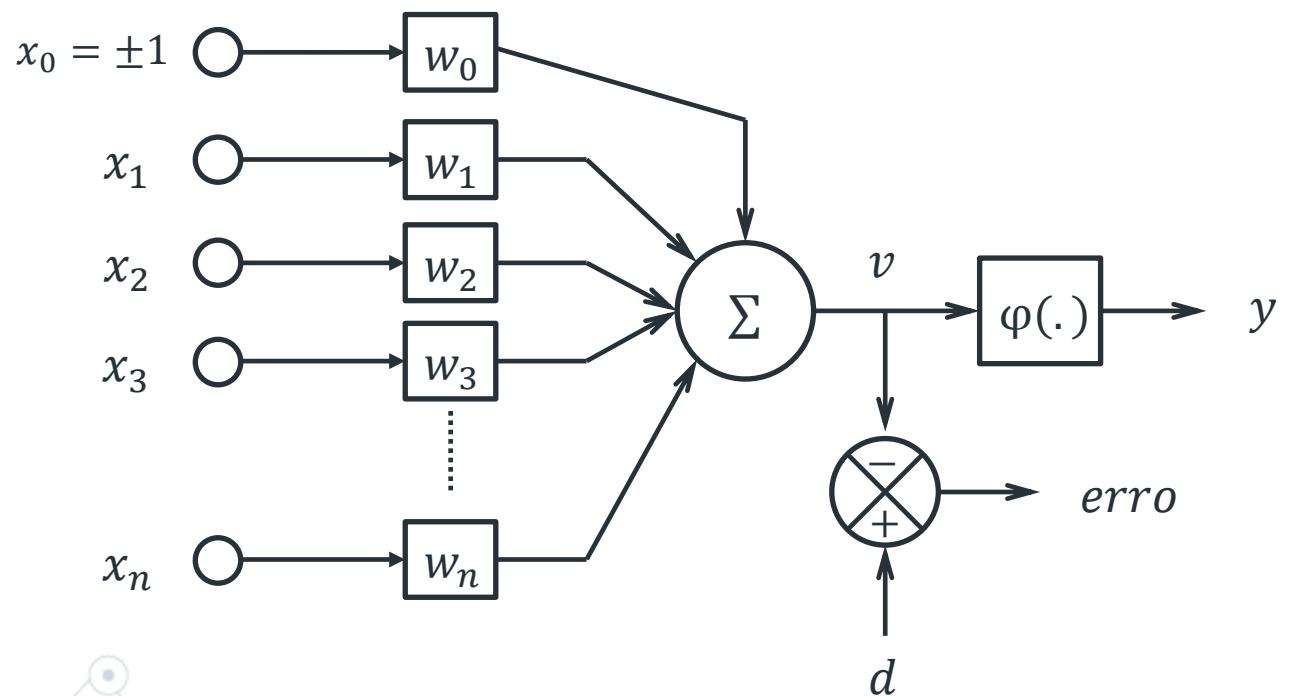


# REDES ADALINE

- *Adaline (Adaptive Linear Neuron)* foi idealizado por Widrow e Hoff em 1960 e sua principal aplicação se destinava a sistemas de chaveamento de circuitos eletrônicos;
- Apesar de ser uma rede simples (**um único neurônio**), promoveu alguns avanços importantes para o progresso da área de redes neurais:
  - Desenvolvimento do **algoritmo de aprendizado regra Delta**;
  - Aplicações em diversos problemas práticos envolvendo processamento de sinais analógicos;
  - Primeiras aplicações industriais de redes neurais artificiais.
- Sua grande contribuição foi a introdução da **Regra Delta**, que hoje é **utilizado para treinamento de redes *Perceptron* de camadas múltiplas**.

# REDES ADALINE

## ARQUITETURA:

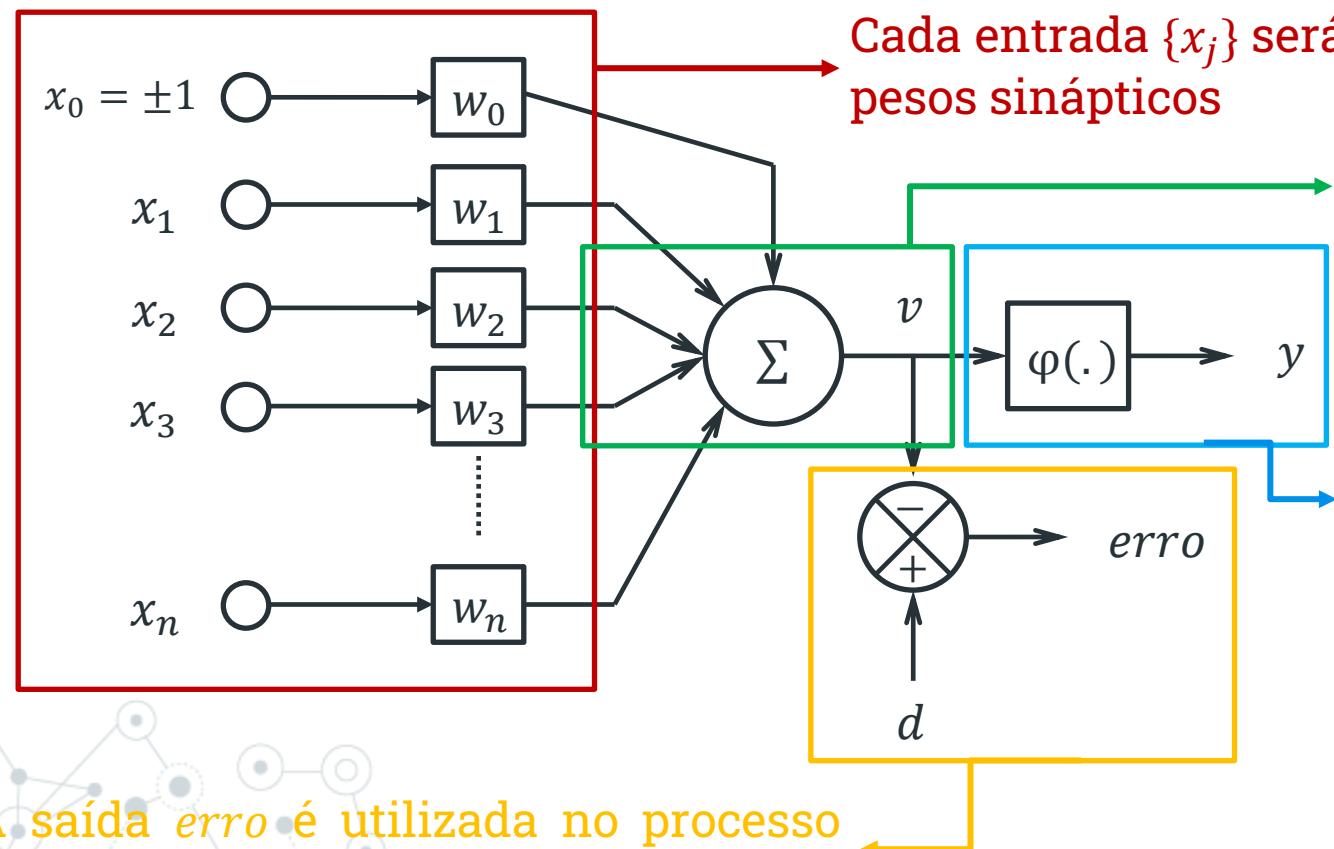


- Uma Camada de Neurônios;
- Sem realimentação (*Feedforward*).
- Similar a Redes Perceptron
- Utilizada para Classificação de Padrões envolvendo apenas 2 classes.

# REDES ADALINE

## ARQUITETURA:

→ Constituída de  $n$  sinais de entrada e somente uma saída (1 neurônio).



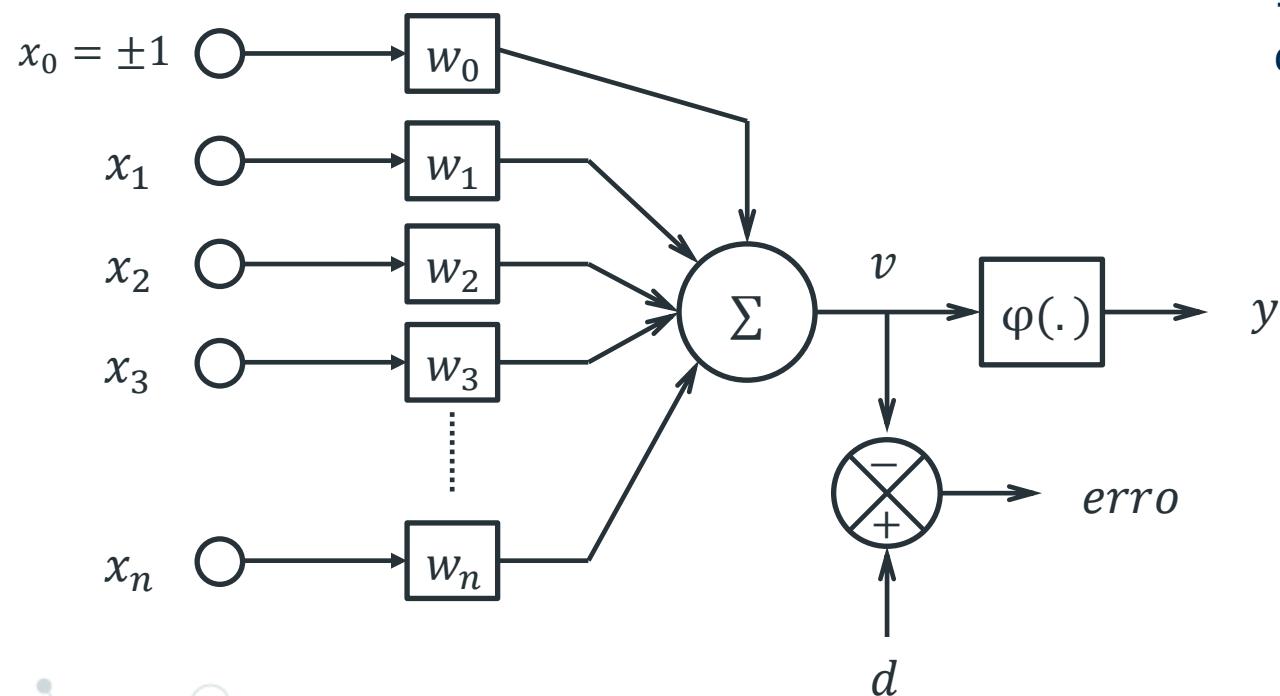
$$v = \sum_{i=1}^n w_i \cdot x_i - \theta \leftrightarrow v = \sum_{i=0}^n w_i \cdot x_i$$

$$y = \varphi(v)$$

$$\text{erro} = (d - v)$$

# REDES ADALINE

## ARQUITETURA:

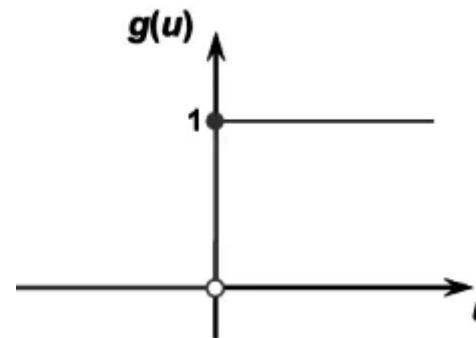


$$v = \sum_{i=1}^n w_i \cdot x_i - \theta \leftrightarrow v = \sum_{i=0}^n w_i \cdot x_i$$

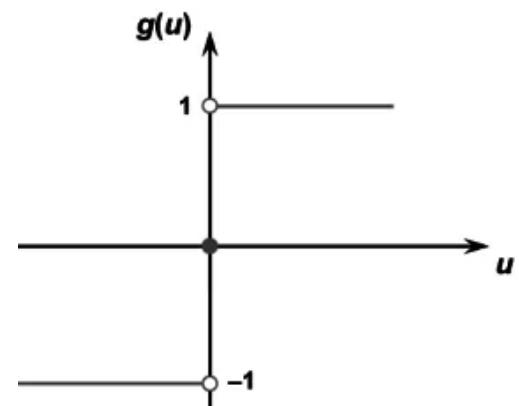
$$y = \varphi(v)$$

$$\text{erro} = (d - v)$$

→ Assim como no **Perceptron**, as funções de ativação do **ADALINE** são a “degrau” ou “degrau bipolar”.



Função Degrau



Função Degrau Bipolar

# REDES ADALINE

## ◎ TREINAMENTO:

- Assim como no **Perceptron**, o ajuste dos pesos e limiar do **ADALINE** é efetuado utilizando processo de treinamento “**Supervisionado**”. Então, para cada amostra dos sinais de entrada se tem a respectiva saída (resposta) desejada.
- A diferença principal entre o **ADALINE** e o **Perceptron** está principalmente na regra de aprendizado utilizada para os ajustes dos pesos e limiar.
- O processo de ajuste dos pesos e limiar do *Adaline* é baseado na **Regra Delta**, também conhecido como algoritmo **LMS (*least mean squares*)** ou método do **Gradiente Descendente**;

# REDES ADALINE

## ◎ TREINAMENTO:

- O processo de ajuste dos pesos e limiar do *Adaline* é baseado na **Regra Delta**, também conhecido como algoritmo **LMS (*least mean squares*)** ou método do **Gradiente Descendente**;
- Assumindo-se  $p$  amostras de treinamento, a ideia básica está em **minimizar a diferença entre a saída desejada  $\{d\}$  e a resposta do combinador linear  $\{v\}$** , considerando todas as amostras;
- O critério utilizado é a **minimização do erro quadrático médio** entre a saída do combinador  $v$  e a saída desejada  $d$  ajustando o vetor de pesos sinápticos  $w = [\theta \ w_1 \ w_2 \ \dots \ w_n]^T$  da rede;

# REDES ADALINE

## ◎ TREINAMENTO:

- O objetivo é encontrar o  $w^*$  ótimo para o qual o erro quadrático  $\{E(w^*)\}$  seja mínimo para todo o conjunto de amostras;

$$E(w^*) \leq E(w), \text{ para } \forall w \in R^{n+1}$$

- A função do erro quadrático médio em relação às  $p$  amostras é definida por:

$$E(w) = \frac{1}{p} \sum_{k=1}^p (d^{(k)} - v)^2 = \frac{1}{p} \sum_{k=1}^p \left( d^{(k)} - \left( \sum_{i=0}^n w_i \cdot x_i^{(k)} \right) \right)^2$$

$$E(w) = \frac{1}{p} \sum_{k=1}^p (d^{(k)} - (w^T * x^{(k)}))^2$$

→ Forma Vetorial

Totaliza o erro quadrático contabilizando-se as  $p$  amostras de treinamento.

Como minimizar a função do erro  $E(w)$ ?

# REDES ADALINE

## REGRA DELTA:

- Consiste de aplicar o operador gradiente em relação ao vetor  $\mathbf{w}$ , tendo-se como objetivo a busca do valor ótimo para o erro quadrático, isto é:

$$\nabla E(\mathbf{w}) = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

- Considerando o valor de  $E(\mathbf{w})$ , tem-se:

$$\nabla E(\mathbf{w}) = \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \sum_{k=1}^p \left( d^{(k)} - (\mathbf{w}^T * \mathbf{x}^{(k)}) \right) \cdot (-\mathbf{x}^{(k)})$$

$$E(\mathbf{w}) = \frac{1}{p} \sum_{k=1}^p \left( d^{(k)} - (\mathbf{w}^T * \mathbf{x}^{(k)}) \right)^2$$



$$\nabla E(\mathbf{w}) = - \sum_{k=1}^p (d^{(k)} - v) \cdot (\mathbf{x}^{(k)})$$

Esta expressão fornece o valor do gradiente do erro em relação a todas as amostras de treinamento.

O vetor gradiente aponta para a direção e sentido de maior crescimento da função de custo (para uma função de ativação linear) → Logo, o ajuste dos pesos deve considerar a mesma direção e o sentido contrário ao do vetor gradiente da função de custo  $E(\mathbf{w})$  porque o objetivo é minimizar o erro quadrático médio.

# REDES ADALINE

## REGRA DELTA:

- Agora, deve-se associar o valor do gradiente ao procedimentos de ajustes do vetor de pesos  $\{w\}$ .
- Assim, a adaptação do vetor de pesos devem ser efetuado na direção oposta àquela do gradiente, pois o objetivo da otimização é minimizar o erro quadrático médio entre  $d$  e  $u$ .

$$\Delta w = -\eta \cdot \nabla E(w)$$

- O ajuste  $\Delta w$  no vetor de pesos sinápticos é dada por:

$$w^{atual} = w^{anterior} + \eta \cdot \sum_{k=1}^p (d^{(k)} - v) \cdot (x^{(k)})$$

- A partir do ajuste  $\Delta w$ , atualiza-se o vetor  $w$ :

$$w^{atual} = w^{anterior} + \eta \cdot (d^{(k)} - v) \cdot (x^{(k)}), \text{ onde } k = 1, \dots, p$$

O ajuste também pode ser realizado após a apresentação de cada  $k$ -ésima amostra de treinamento:

# REDES ADALINE

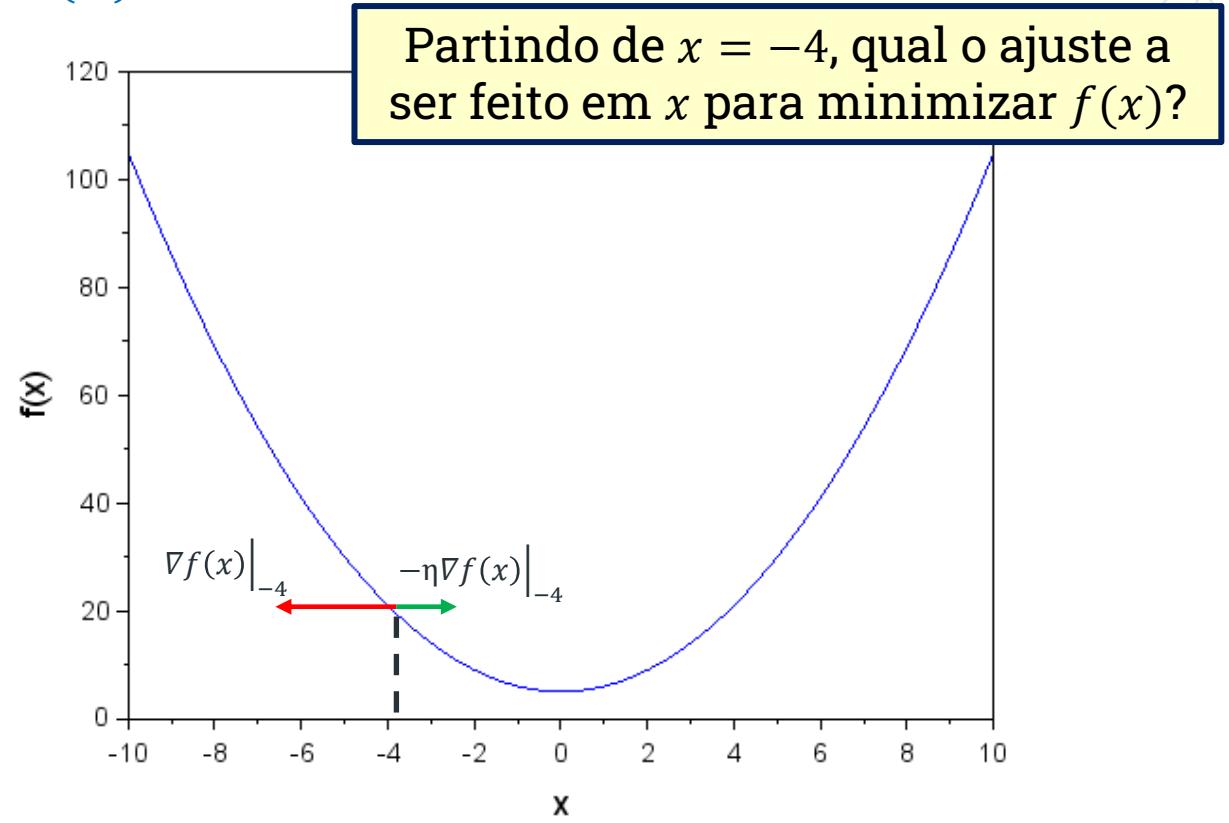
## GRADIENTE:

- Como minimizar a função do erro  $E(w)$ ?

$$f(x) = x^2 + 5 \rightarrow \nabla f(x) = 2x$$

$$\begin{cases} x = -4 \rightarrow \nabla f(x) = -8 \\ x = -3 \rightarrow \nabla f(x) = -6 \\ x = -2 \rightarrow \nabla f(x) = -4 \\ x = -1 \rightarrow \nabla f(x) = -2 \\ x = 0 \rightarrow \nabla f(x) = 0 \\ x = +1 \rightarrow \nabla f(x) = +2 \\ x = +2 \rightarrow \nabla f(x) = +4 \\ x = +3 \rightarrow \nabla f(x) = +6 \\ x = +4 \rightarrow \nabla f(x) = +8 \end{cases}$$

$$\nabla f(x^*) = 0 \rightarrow x^* = (0,0)$$



# REDES ADALINE

## GRADIENTE:

- Com a aplicação dos passos iterativos anteriores, o vetor  $\mathbf{w}$  caminhará em direção ao seu valor ótimo  $\{\mathbf{w}^*\}$ .
- Para se detectar o alcance de  $\mathbf{w}^*$ , a fim de cessar o processo de convergência, há a necessidade de se estabelecer um critério de parada.
- Para tanto, utiliza-se o valor do erro quadrático médio  $\{E_{qm}(\mathbf{w})\}$  em relação a todas as amostras de treinamento, sendo este definido por:

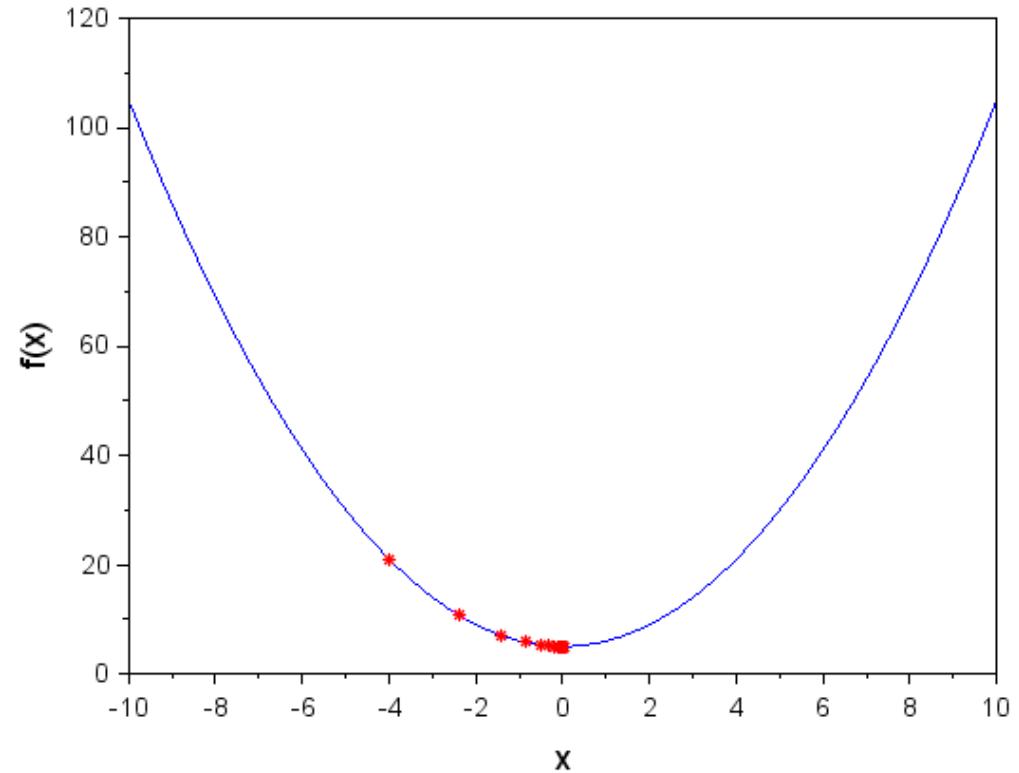
$$E_{qm}(\mathbf{w}) = \frac{1}{p} \sum_{k=1}^p (d^{(k)} - v)^2$$

- O algoritmo converge quando o erro quadrático médio entre duas épocas sucessivas for bem pequeno, isto é:

$$|E_{qm}(\mathbf{w}^{\text{atual}}) - E_{qm}(\mathbf{w}^{\text{anterior}})| \leq \varepsilon$$

→ onde  $\varepsilon$  é a precisão requerida para o processo de convergência

$$x(k+1) = x + \eta \cdot \nabla f(x(k)) \rightarrow f(x(k+1)) < f(x(k))$$



# REDES ADALINE

## ◎ REGRA DELTA:

- Assim como no **Perceptron**, a taxa de aprendizagem  $\{\eta\}$  exprime o quanto rápido o processo de treinamento da rede estará rumando em direção ao ponto de minimização da função de erro quadrático médio;
- Normalmente adotam-se valores pertencentes ao intervalo compreendido em  $0 < \eta < 1$ , comumente expressos em potências de 10. Exemplos:
  - $0,1 \rightarrow 10^{-1}$  ou  $1E-1$ , que significa  $1,0 \times 10^{-1}$
  - $0,01 \rightarrow 10^{-2}$  ou  $1E-2$ , que significa  $1,0 \times 10^{-2}$
  - $0,001 \rightarrow 10^{-3}$  ou  $1E-3$ , que significa  $1,0 \times 10^{-3}$
  - E assim em diante.

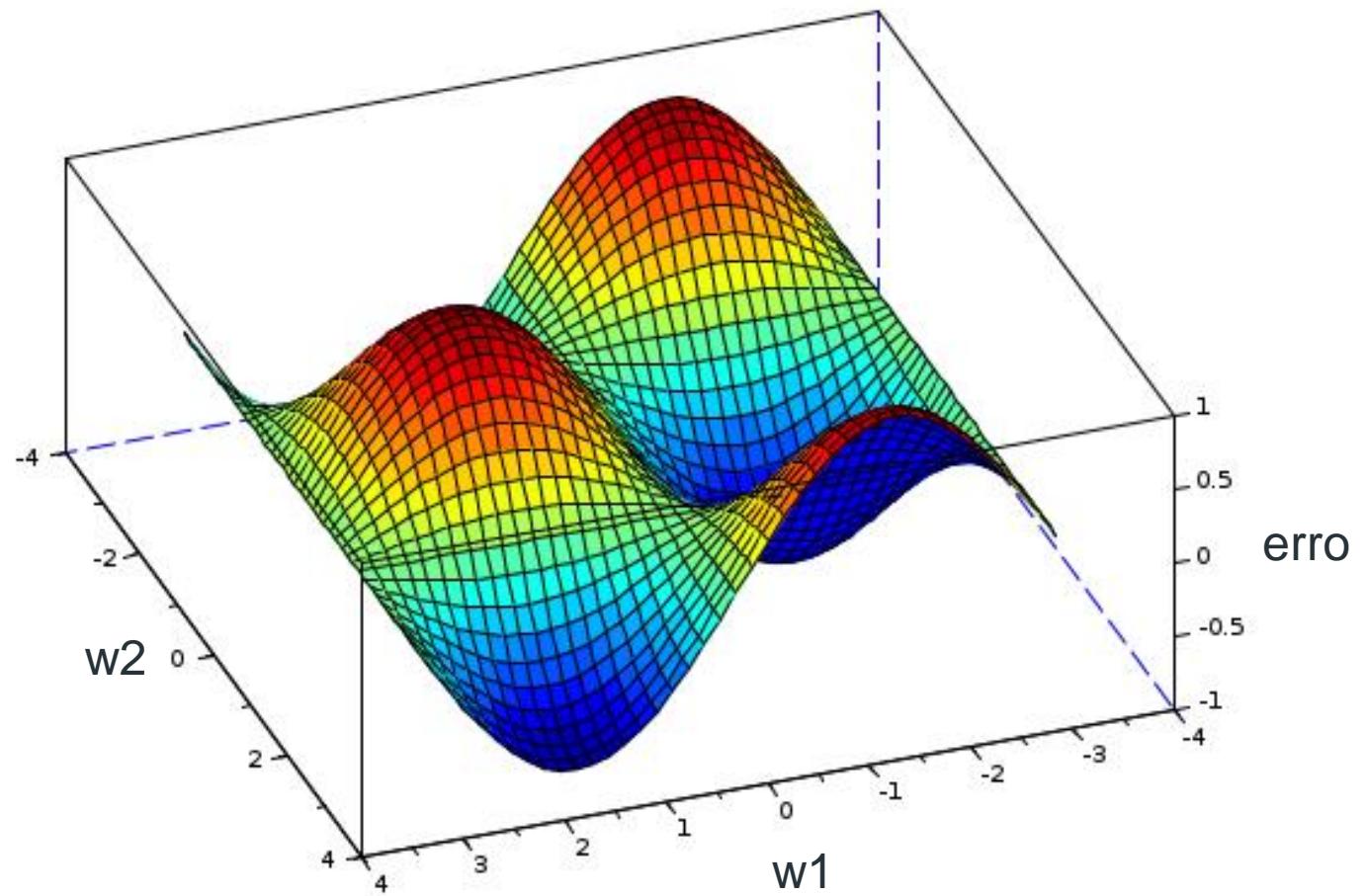
# REDES ADALINE

## ○ SUPERFÍCIES DE ERRO:

- O conjunto dos  $m + 1$  pesos a serem ajustados em uma rede neural pode ser visto como um ponto em um espaço  $(m + 1)$ -dimensional, conhecido como **espaço de pesos**;
- Pode-se imaginar que cada conjunto de pesos apresenta um valor associado de erro para cada amostra de entrada e também para todo o **conjunto de treinamento**;
- Os **valores de erro para todos os conjuntos possíveis de pesos** definem uma superfície no espaço de pesos – a **superfície de erro**;

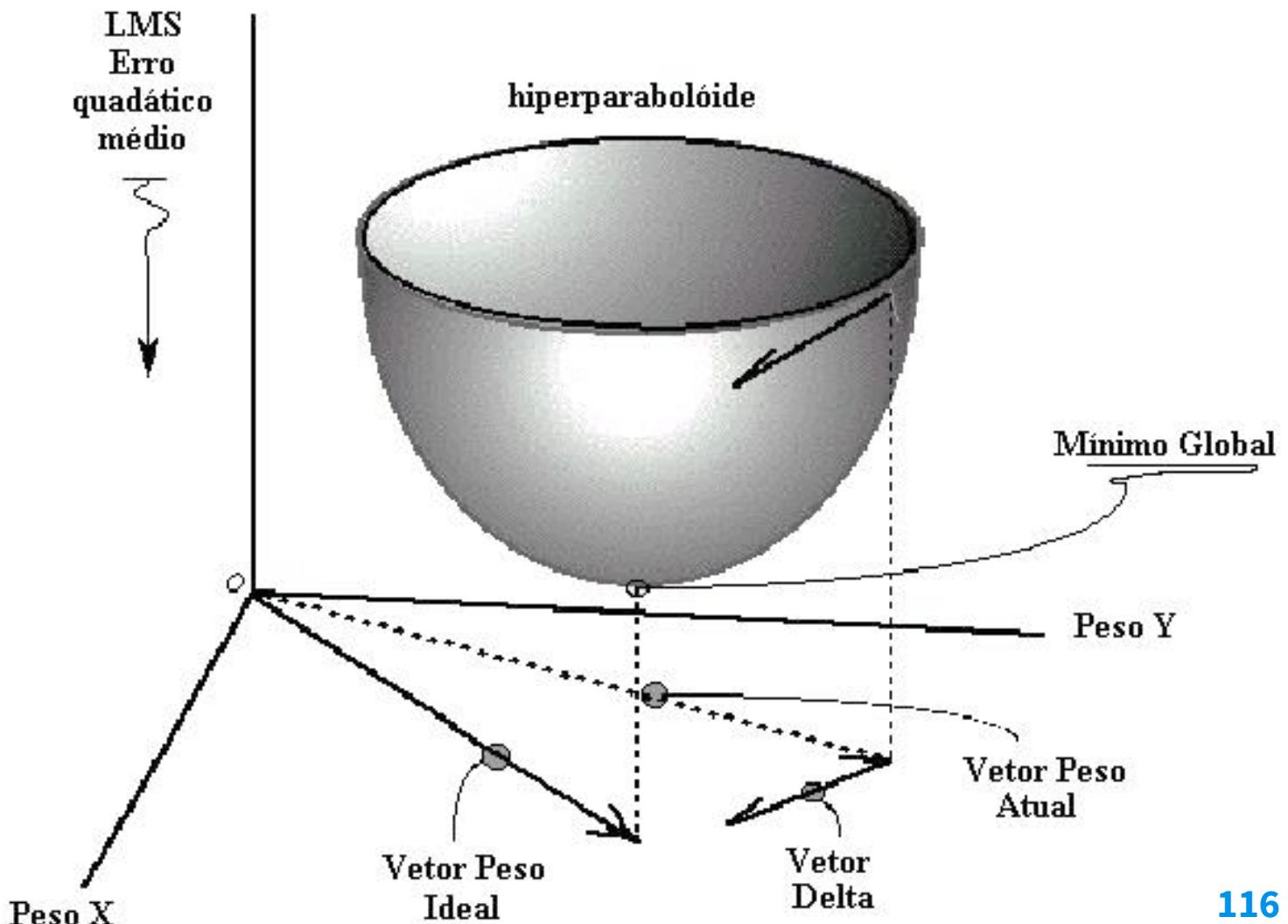
# REDES ADALINE

## SUPERFÍCIES DE ERRO:



# REDES ADALINE

## SUPERFÍCIES DE ERRO:



# REDES ADALINE

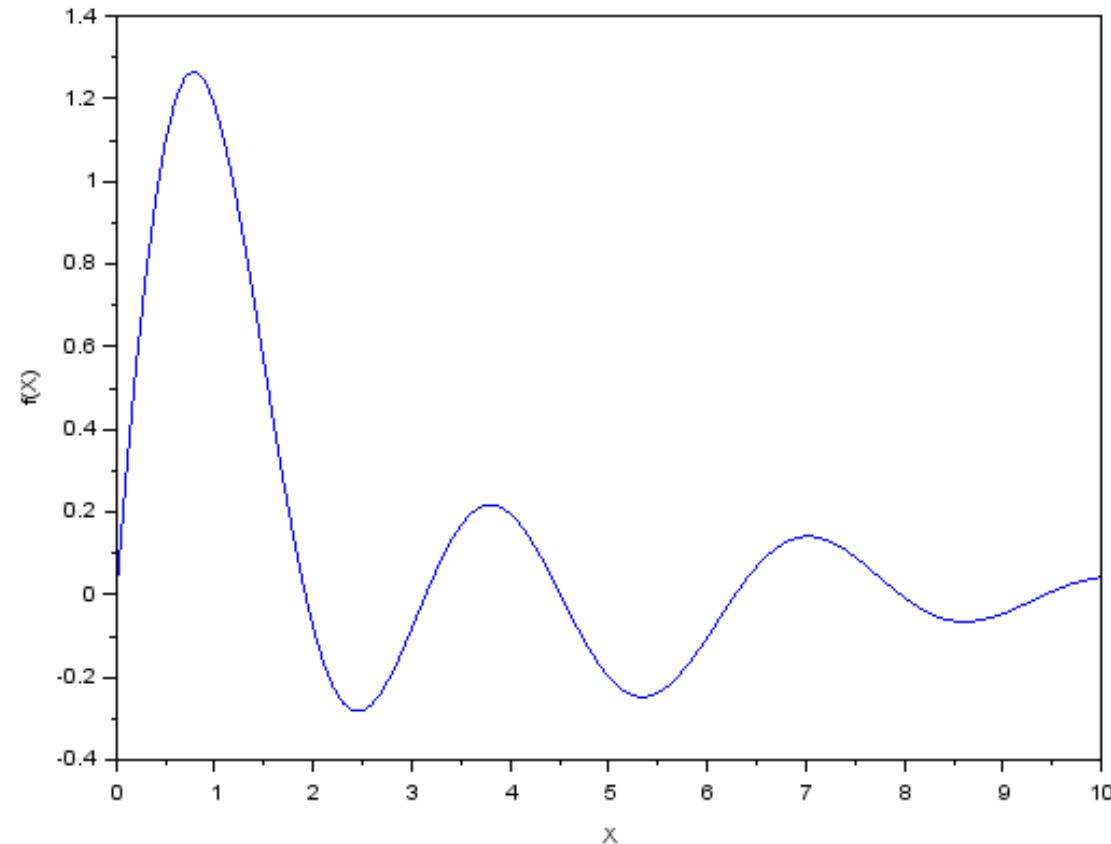
## SUPERFÍCIES DE ERRO:

- O algoritmo de treinamento baseado na regra Delta, que é um **algoritmo supervisionado**, opera com base na **minimização de uma função de custo**, que neste caso é baseada no erro entre o resultado do combinador linear e as saídas desejadas.
- **Observações:**
  - A superfície de erro possui uma **grande quantidade de mínimos locais**;
  - Pode-se encontrar situações onde **o método do gradiente não tem solução (descontinuidades)**;
  - Outros métodos de busca podem ser utilizados.

# REDES ADALINE

- **SOBRE A TAXA DE APRENDIZADO:**

- Como uma escolha errada da taxa de aprendizagem pode afetar o treinamento?

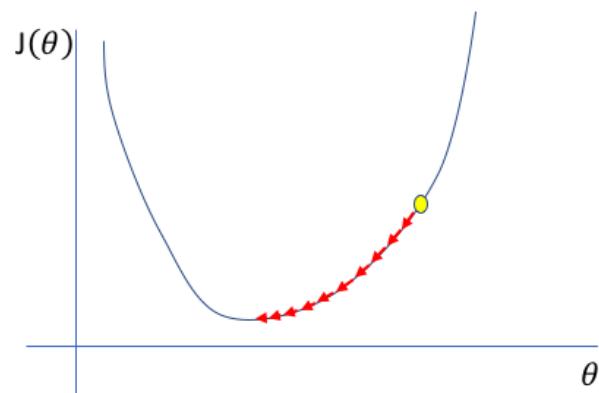


# REDES ADALINE

- SOBRE A TAXA DE APRENDIZADO:

$\eta$  muito baixa

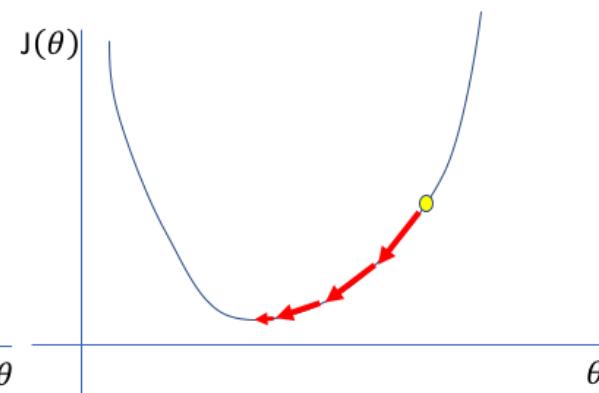
Too low



A small learning rate requires many updates before reaching the minimum point

$\eta$  correta

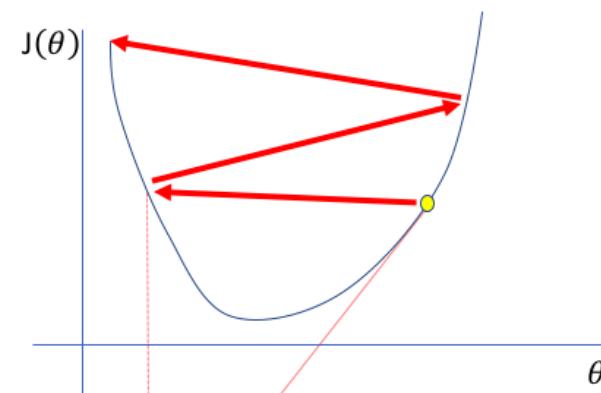
Just right



The optimal learning rate swiftly reaches the minimum point

$\eta$  muito alta

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

# TREINAMENTO ADALINE

- 1) Obter conjunto de amostras de treinamento  $\{x^{(k)}\}$ ;
- 2) Associar a saída desejada  $\{d^{(k)}\}$  para cada amostra obtida;
- 3) Iniciar o vetor  $w$  com valores aleatórios pequenos;
- 4) Especificar a taxa de aprendizagem  $\{\eta\}$  e precisão requerida  $\{\varepsilon\}$ ;
- 5) Iniciar o contador de número de épocas  $\{\text{épocas} \leftarrow 0\}$ ;
- 6) **Repetir as instruções:**

$$6.1) E_{qm}^{anterior} \leftarrow E_{qm}(w)$$

6.2) Para todos pares de treinamento  $\{x^{(k)}, d^{(k)}\}$ , faça:

$$\begin{cases} 6.2.1) v \leftarrow w^T * x^k; \\ 6.2.2) w \leftarrow w + \eta * (d^{(k)} - v) * x^{(k)}; \end{cases}$$

6.3)  $\text{época} \leftarrow \text{época} + 1$ ;

$$6.4) E_{qm}^{atual} \leftarrow E_{qm}(w);$$

**Até que:**  $|E_{qm}^{atual} - E_{qm}^{anterior}| \leq \varepsilon$

# ALGORITMO EQM (Erro Quadrático Médio)

- 1) Obter a quantidade de padrões de treinamento  $\{p\}$ ;
- 2) Iniciar a variável  $E_{qm}$  com valor zero  $\{E_{qm} \leftarrow 0\}$ ;
- 3) Para todos pares de treinamento  $\{x^{(k)}, d^{(k)}\}$ , faça:
  - 3.1)  $v \leftarrow w^T * x^k$ ;
  - 3.2)  $E_{qm} \leftarrow E_{qm} + (d^{(k)} - v)^2$ ;
  - 3.3)  $E_{qm} \leftarrow \frac{E_{qm}}{p}$

# FUNCIONAMENTO ADALINE

- 1) Obter a amostra a ser classificada  $\{x\}$ ;
- 2) Utilizar o vetor  $w$  ajustado durante o treinamento;
- 3) Executar as seguintes instruções:

3.1)  $v \leftarrow w^T * x$ ; → Potencial de Ativação

3.2)  $y \leftarrow \text{degrau}(v)$ ; (sign no Matlab) → Função de Ativação

3.3) Se  $y == -1$

3.3.1) Então: amostra  $x \in \{\text{Classe } A\}$

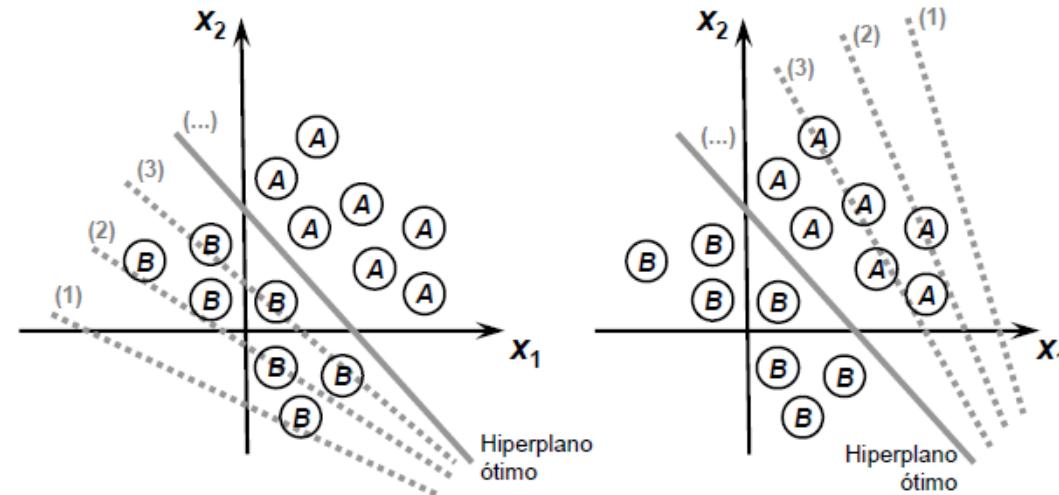
3.4) Se  $y == 1$

3.4.1) Então: amostra  $x \in \{\text{Classe } B\}$

CLASSIFICAÇÃO

# CONVERGÊNCIA ADALINE

- Processo de treinamento do **ADALINE** tende a mover continuamente o seu vetor de pesos, tendo-se o intuito de **minimizar o erro quadrático** em relação a todas as amostras disponíveis para o aprendizado.



Convergência das redes ADALINE rumo à estabilização, considerando-se duas entradas  $\{x_1 \text{ e } x_2\}$ .

Observa-se que nestas duas situações, cujos vetores de pesos iniciais  $w^{(0)}$  foram iniciados em posições distintas, o processo de convergência encaminha o hiperplano sempre para o mesmo local.

Nesses casos, o valor correspondente a  $w^*$  é sempre aquele que minimiza a função erro quadrático.

# REDES ADALINE

## EXEMPLO TREINAMENTO ADALINE:

- Supondo um problema a ser mapeado pelo *Adaline* com duas entradas  $\{x_1, x_2\}$ ;
- Para um conjunto de quatro amostras de treinamento constituídas dos seguintes valores:

$$\Omega^{(x)} = \{[2.0 \ 3.5]; [6.8 \ 5.3]; [2.0 \ 2.5]; [8.1 \ 4.2]\}.$$

- Considerando-se ainda que os respectivos valores de saída para cada uma das amostras seja dado por:

$$\Omega^{(d)} = \{[-1]; [1]; [-1]; [1]\}.$$

- Escolhendo aleatoriamente os pesos sinápticos iniciais:

$$w = \{0.36; 0.29; 0.57\}.$$

$$\Omega^{(x)} = \begin{matrix} & x^{(1)} & x^{(2)} & x^{(3)} & x^{(4)} \\ \begin{matrix} x_0 \\ x_1 \\ x_2 \end{matrix} & \left[ \begin{matrix} -1 & -1 & -1 & -1 \\ 2.0 & 6.8 & 2.0 & 8.1 \\ 3.5 & 5.3 & 2.5 & 4.2 \end{matrix} \right] \end{matrix} \text{Características}$$

$$\Omega^{(d)} = \begin{matrix} & d^{(1)} & d^{(2)} & d^{(3)} & d^{(4)} \\ \{ & -1 & 1 & -1 & 1 \} & \} \end{matrix} \text{Saídas Esperadas}$$

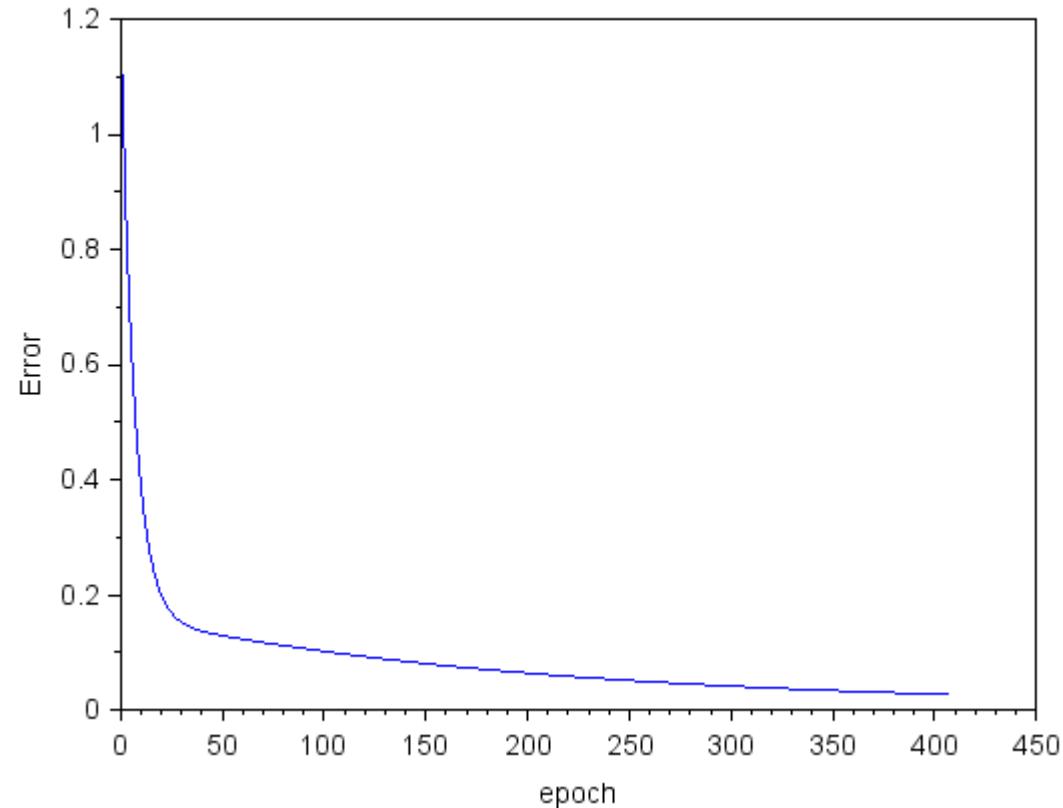
-1 → Classe A - o  
+1 → Classe B - x

# REDES ADALINE

## Exemplo Treinamento Adaline:

- Após 407 épocas de treinamento:

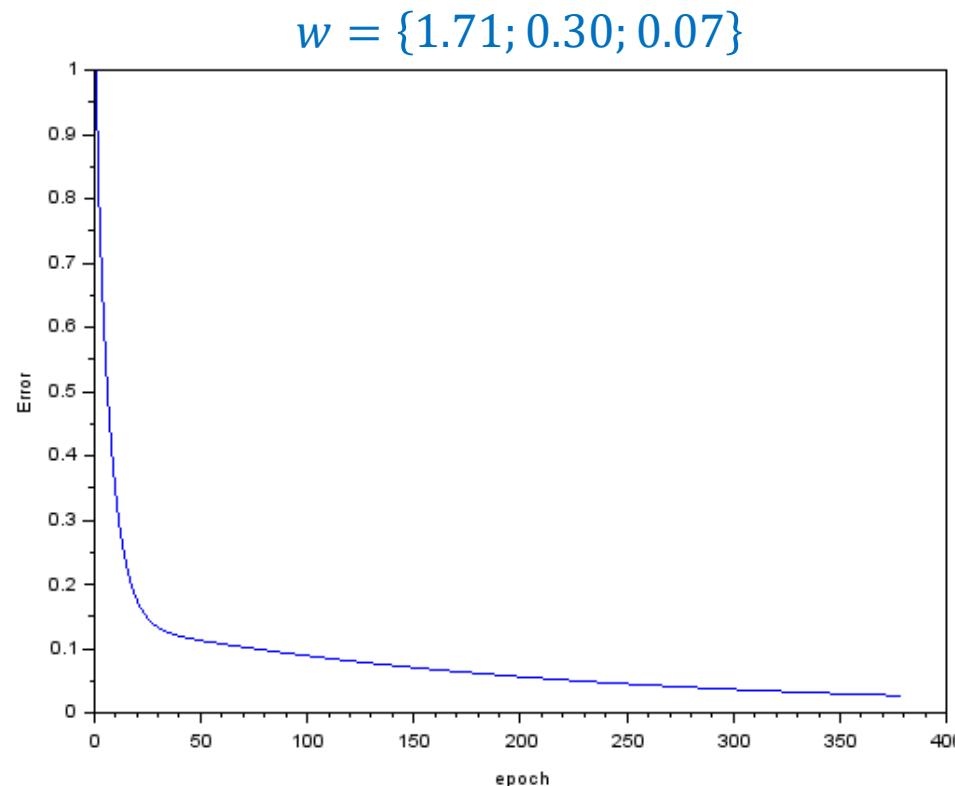
$$w = \{1.71; 0.30; 0.07\}$$



# REDES ADALINE

## EXEMPLO TREINAMENTO ADALINE:

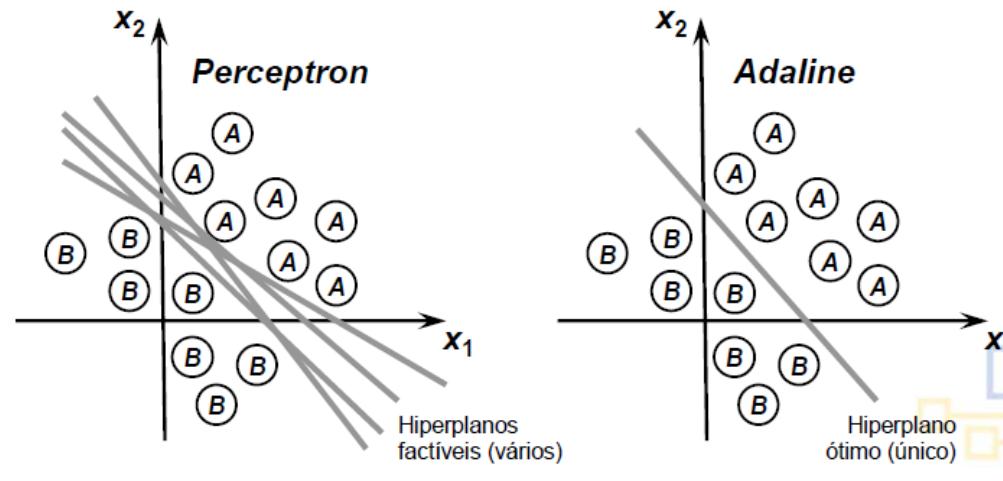
- Usando o mesmo problema anterior porém escolhendo outros valores, aleatoriamente, para os pesos sinápticos iniciais, obtém-se após 379 épocas de treinamento:



- Constata-se então que a **curva do erro quadrático médio** é sempre descendente.
- Esta decresce na medida em que as épocas de treinamento vão sendo executadas.
- Finalmente, estabiliza-se num valor constante quando **o ponto de mínimo da função erro quadrático médio** for alcançado.

# REDES PERCEPTRON x ADALINE

- ◎ **ADALINE** → Treinamento usando **Regra Delta** (Minimização do erro em relação a todas as amostras de treinamento).
  - Independentemente dos valores iniciais atribuídos a  $w$ , o hiperplano de separabilidade (após convergência) sempre será o mesmo.
  
- ◎ **PERCEPTRON** → Treinamento usando **Regra de Hebb** (Avaliação das respostas produzidas após apresentação de cada amostra de treinamento).
  - Quaisquer **hiperplanos** posicionados dentro da faixa de separabilidade entre as classes são considerados soluções factíveis.

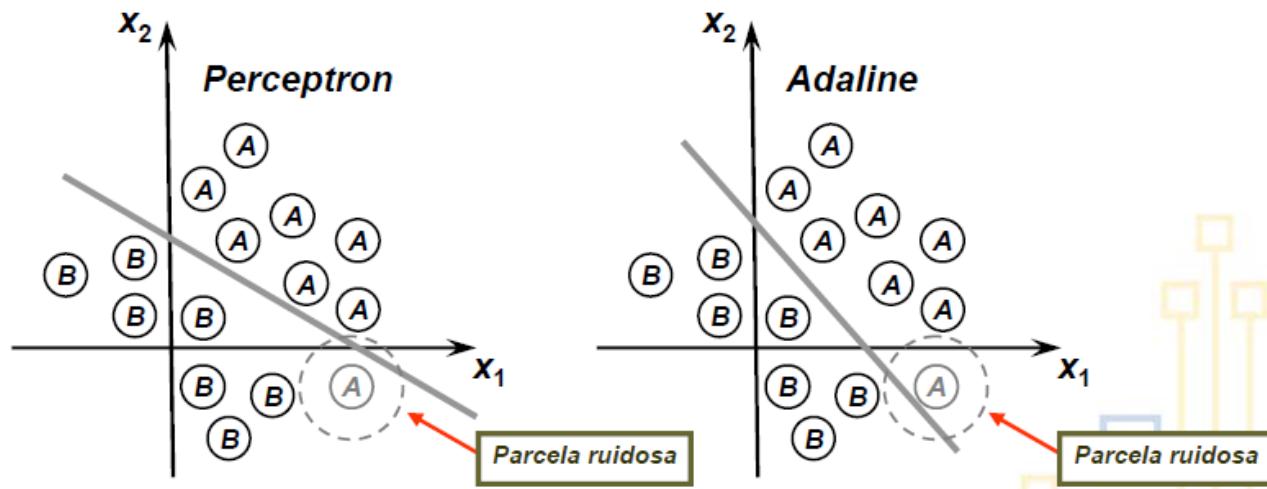


# REDES PERCEPTRON x ADALINE

- ◎ **ADALINE** → A inclinação do hiperplano é ajustada por intermédio **do método dos mínimos quadrados dos erros (processo otimizado)**.
- **Maior tendência** de imunidade a eventuais ruídos que podem afetar o processo em que a mesma está mapeando.
- ◎ **PERCEPTRON** → O hiperplano que separa as classes pode ter **infinitas disposições**, pois sua configuração final é fortemente do valores inicial de  $\mathbf{w}$ .
- **Menor tendência** de imunidade a eventuais ruídos que podem afetar o processo em que a mesma está mapeando.

**ADALINE** → probabilidade menor de classificar a amostra incorretamente.

**PERCEPTRON** → probabilidade maior de classificar a amostra incorretamente.



# REDES ADALINE

## OBSERVAÇÕES:

- O processo de treinamento da *Adaline* tende a mover o vetor de pesos sinápticos visando **diminuir o erro quadrático médio em relação a todas as amostras de treinamento**;
- O processo de convergência da rede caminha o hiperplano de separação sempre em direção à **fronteira de separabilidade ótima mesmo começando com vetores iniciais distintos**;
- A curva do erro quadrático médio para o *Adaline* é **sempre descendente**, à medida que as épocas de treinamento são executadas, estabilizando-se num valor constante quando o ponto de mínimo da função de erro quadrático médio é alcançado;

# REDES ADALINE

## ◎ OBSERVAÇÕES:

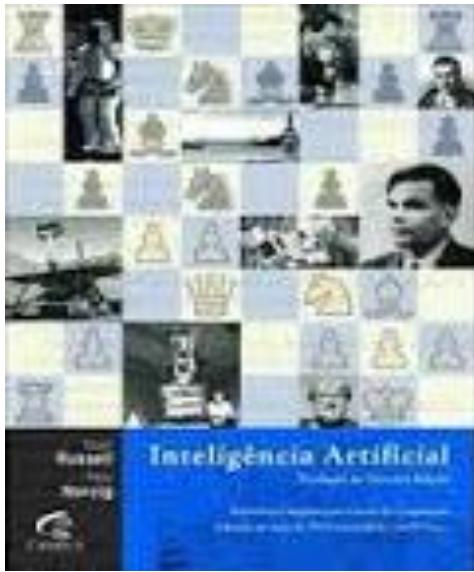
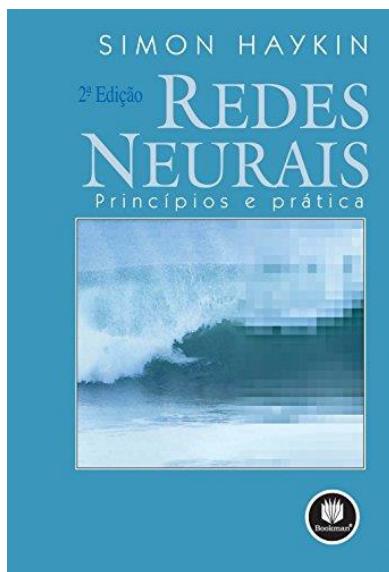
- No *Adaline*, o processo de treinamento busca o hiperplano de separabilidade **ótimo** enquanto no *Perceptron*, **qualquer** hiperplano dentro da faixa de separabilidade é considerado uma solução;
- O *Adaline* ajusta a inclinação do hiperplano através do método dos mínimos quadrados dos erros (*LMS – least mean squares*);
- Por atingir o hiperplano ótimo para quaisquer valores atribuídos inicialmente a seus pesos, o *Adaline* é uma rede neural com maior **imunidade a ruídos**.

# REDES ADALINE

## ◎ OBSERVAÇÕES:

- **O valor da taxa de aprendizagem  $\{\eta\}$  deve ser cuidadosamente especificado** para evitar instabilidades em torno do ponto mínimo e também evitar um processo de convergência extremamente lento;
- O número de épocas de treinamento depende dos pesos sinápticos iniciais atribuídos  $\{w\}$  assim como do valor assumido para a taxa de aprendizagem  $\{\eta\}$ ;
- O desempenho do treinamento do *Adaline* pode ser melhorado por intermédio da **normalização** dos sinais de entrada frente ao domínio apresentado;

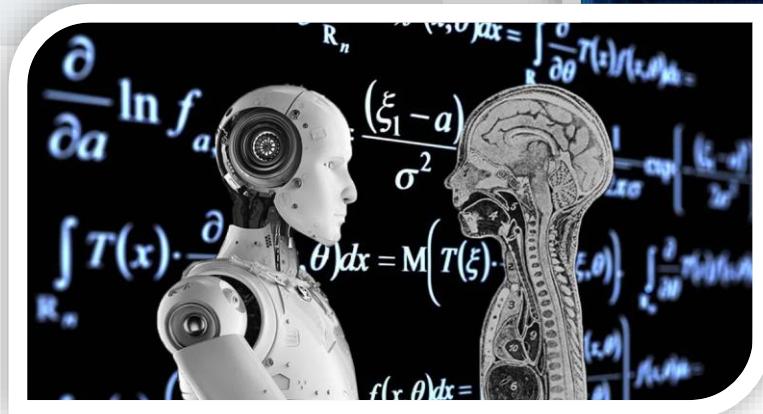
# REFERÊNCIAS



- HAYKIN, Simon S.; ENGEL, Paulo Martins (Paulo Martins Engel), Redes neurais: Princípios e práticas. 2 ed. São Paulo, SP: Editora Bookman, 2001, 900 p. ISBN 978-85-7307-718-6.
- RUSSELL, Stuart; NORVIG, Peter (Peter Norvig); SOUZA, Vandenberg Dantas De, Inteligência artificial. Rio de Janeiro, RJ: Editora Campus, 2004 - 2013, ISBN 978-85-352-1177-1 / 978-85-352-3701-6.
- SILVA, Ivan Nunes da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade, Redes neurais artificiais: para engenharia e ciências aplicadas - curso prático. São Paulo, SP: Editora Artliber, 2010, 399 p. ISBN 978-85-88098-53-4.

# REDES PERCEPTRON MULTICAMADAS

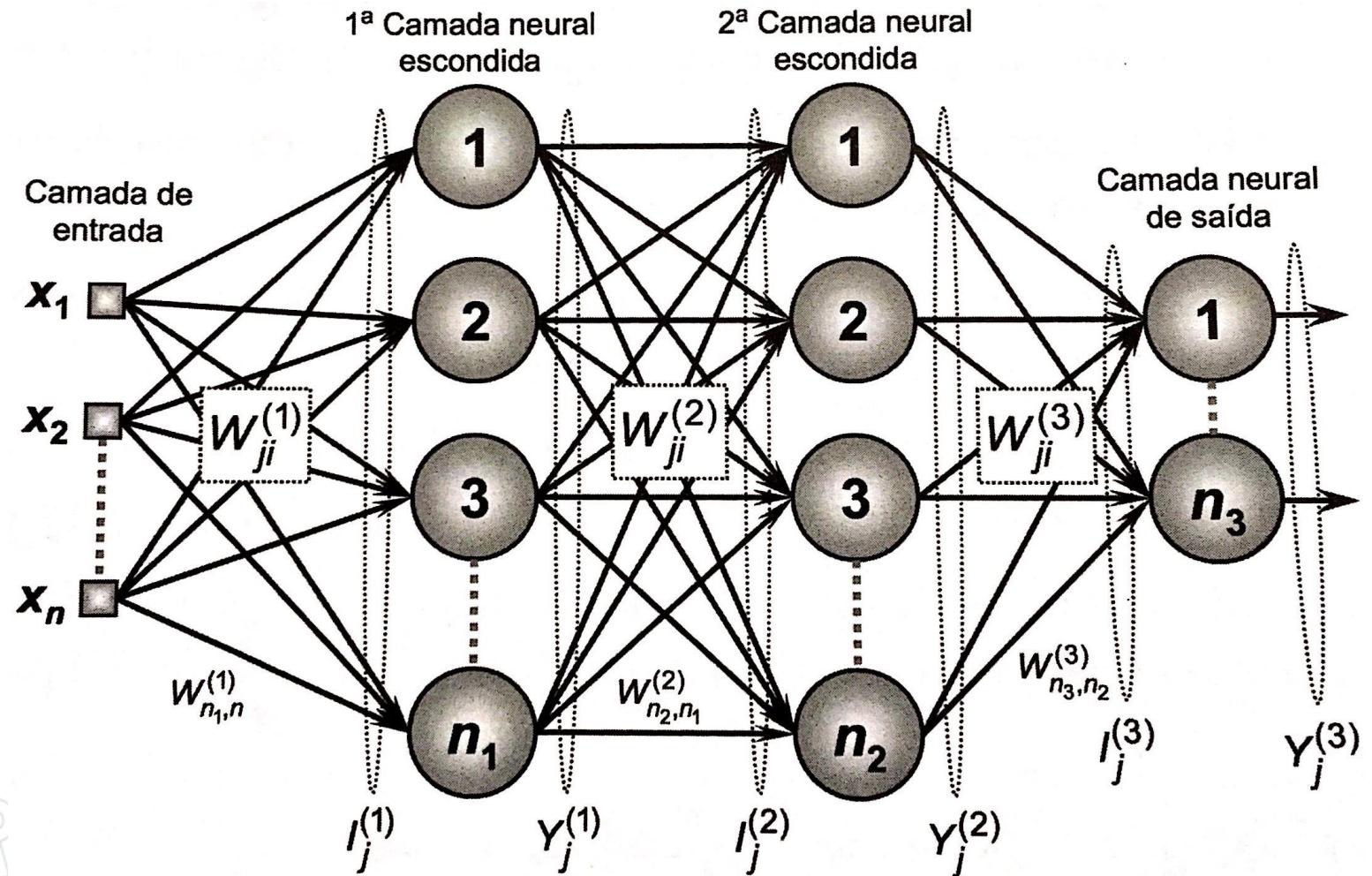
- As **Redes Neurais Artificiais (RNAs)** são baseadas em **modelos abstratos** de como pensamos que o cérebro (e os neurônios) funciona.



# PERCEPTRON MULTICAMADAS

- Caracteriza-se pela presença de pelo menos uma **camada intermediária (escondida)**;
- **Podem ser aplicadas em diversos tipos de problemas:**
  - Aproximação universal de funções;
  - Reconhecimento de padrões;
  - Identificação e controle de processos;
  - Previsão de séries temporais;
  - Otimização de sistemas.
- Possui arquitetura ***feedforward*** de camadas múltiplas;
- Tornou-se famosa após a apresentação do algoritmo ***backpropagation***, apresentado por Rumelhart (1986).

# PERCEPTRON MULTICAMADAS



# PERCEPTRON MULTICAMADAS

- **PRINCÍPIO DE FUNCIONAMENTO:**

- Cada uma das entradas da rede será propagada em direção à camada neural de saída;
- As saídas dos neurônios da primeira camada escondida serão as entradas dos neurônios da segunda camada escondida;
- As saídas da última camada escondida serão as respectivas entradas da camada de saída;
- A propagação dos sinais de entrada ocorre sempre em um único sentido: **entrada para saída (*feedforward*)**;
- Além da presença de camadas escondidas, a rede MLP pode apresentar uma camada de saída composta por vários neurônios, sendo um neurônio para cada processo a ser mapeado;

# PERCEPTRON MULTICAMADAS

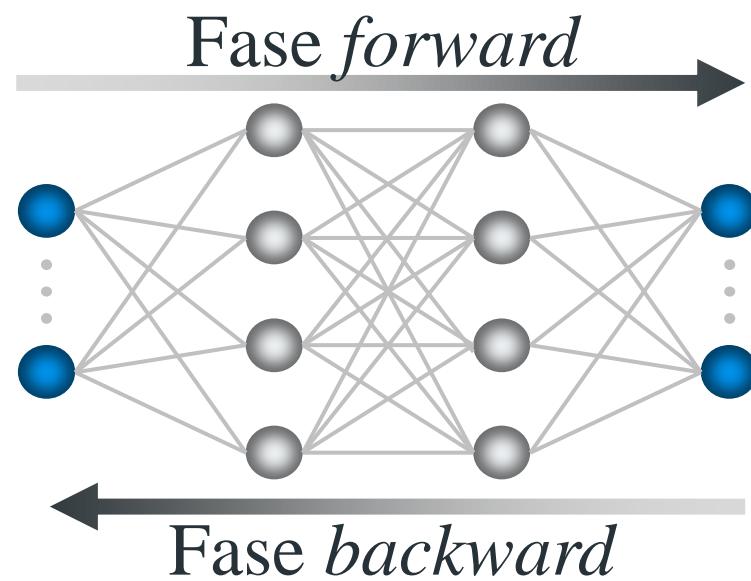
## ○ PRINCÍPIO DE FUNCIONAMENTO:

- O conhecimento relacionado ao comportamento entrada/saída do sistema será distribuído por todos os neurônios da rede MLP;
- A definição da configuração topológica de uma rede MLP depende de uma série de fatores:
  - Disposição espacial das amostras de treinamento;
  - Valores iniciais atribuídos ao parâmetros de treinamento;
  - Valores iniciais atribuídos às matrizes de pesos sinápticos.
- O ajuste dos pesos e do limiar de cada um dos neurônios é efetuado utilizando-se um processo de treinamento supervisionado;
- O algoritmos de aprendizado utilizado é denominado *backpropagation* (retropropagação);

# PERCEPTRON MULTICAMADAS

- **PROCESSO DE TREINAMENTO:**

- A regra de aprendizagem baseada na correção do erro pelo método do gradiente (*LMS – Least Mean Squares*);
- O treinamento é feito em duas fases: *forward* (cálculo do erro) e *backward* (correção dos pesos sinápticos);



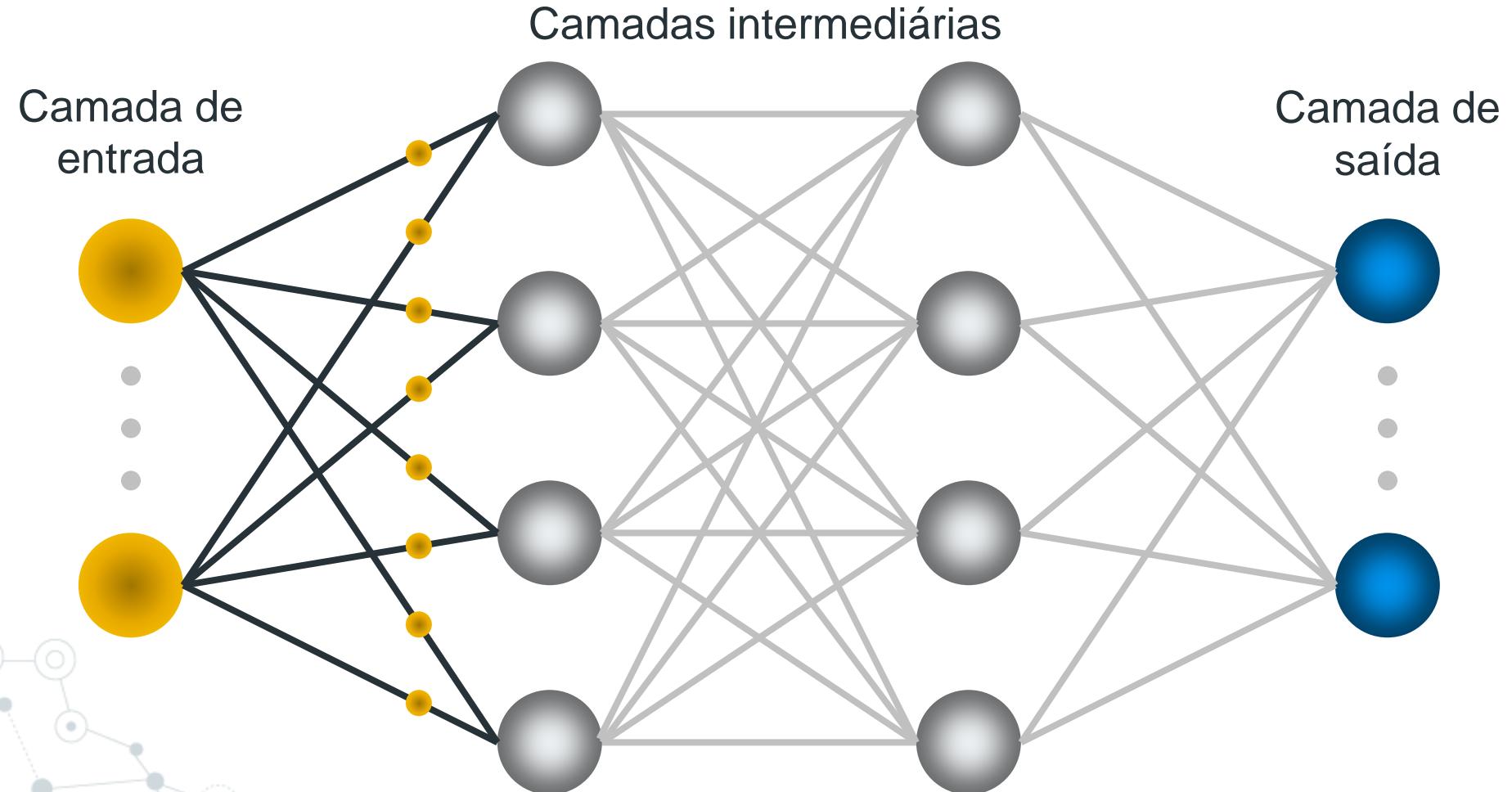
# ALGORITMO BACKPROPAGATION

- Durante o **treinamento com o algoritmo *backpropagation***, a rede opera em uma sequência de dois passos:
  - 1) Um padrão é apresentado à camada de entrada da rede. A atividade resultante flui através da rede, camada por camada, até que a **resposta** seja produzida pela camada de saída.
  - 2) A saída obtida é comparada à saída desejada para esse padrão particular. Se esta não estiver correta, o **erro** é calculado. O erro é propagado a partir da camada de saída até a camada de entrada, e os pesos das conexões das unidades das camadas internas vão sendo modificados conforme o erro é retropropagado.

# ALGORITMO BACKPROPAGATION

- FASE *Forward*:

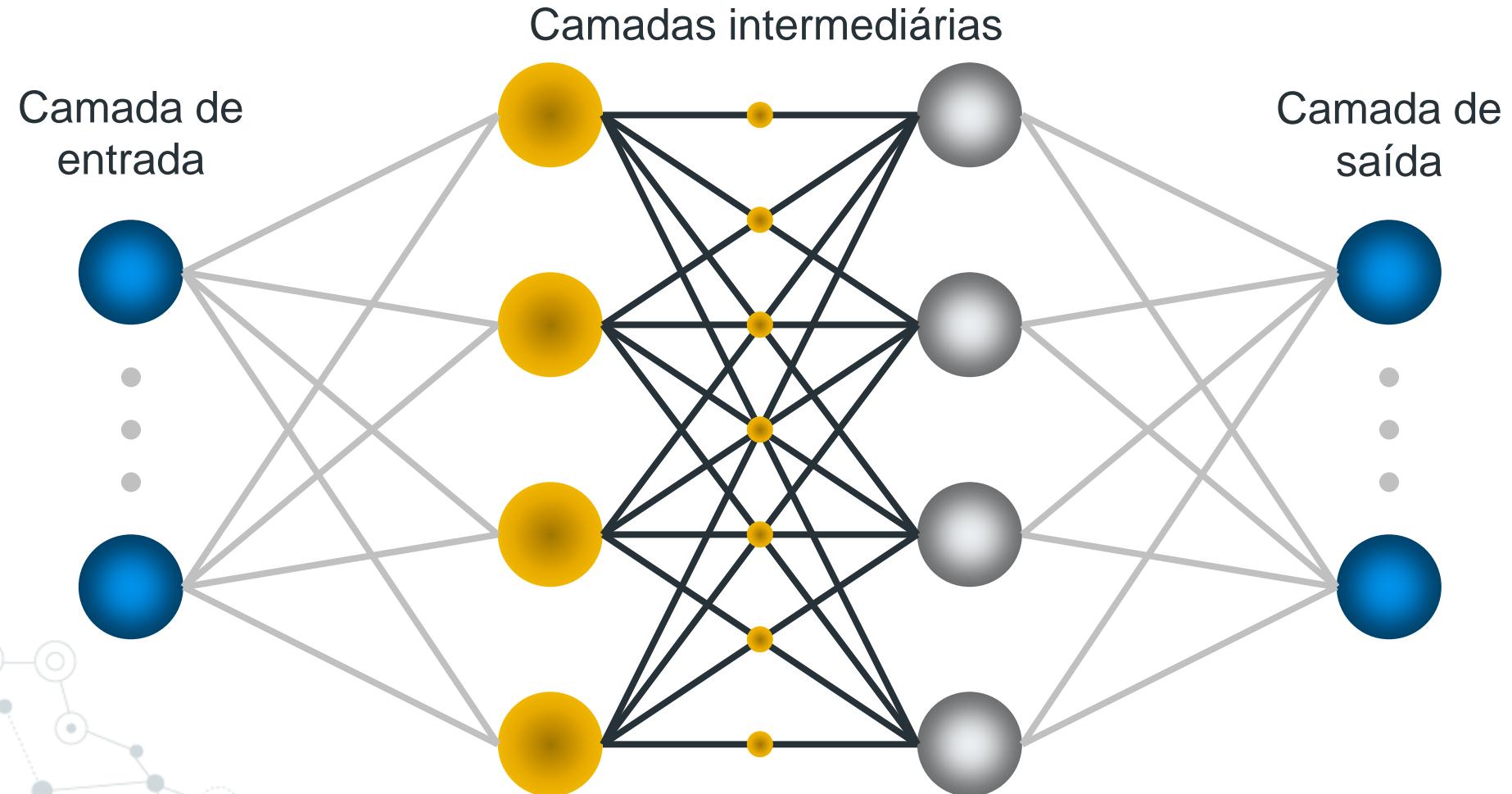
Entrada é apresentada à primeira camada da rede e é propagado em direção às saídas.



# ALGORITMO BACKPROPAGATION

- FASE *Forward*:

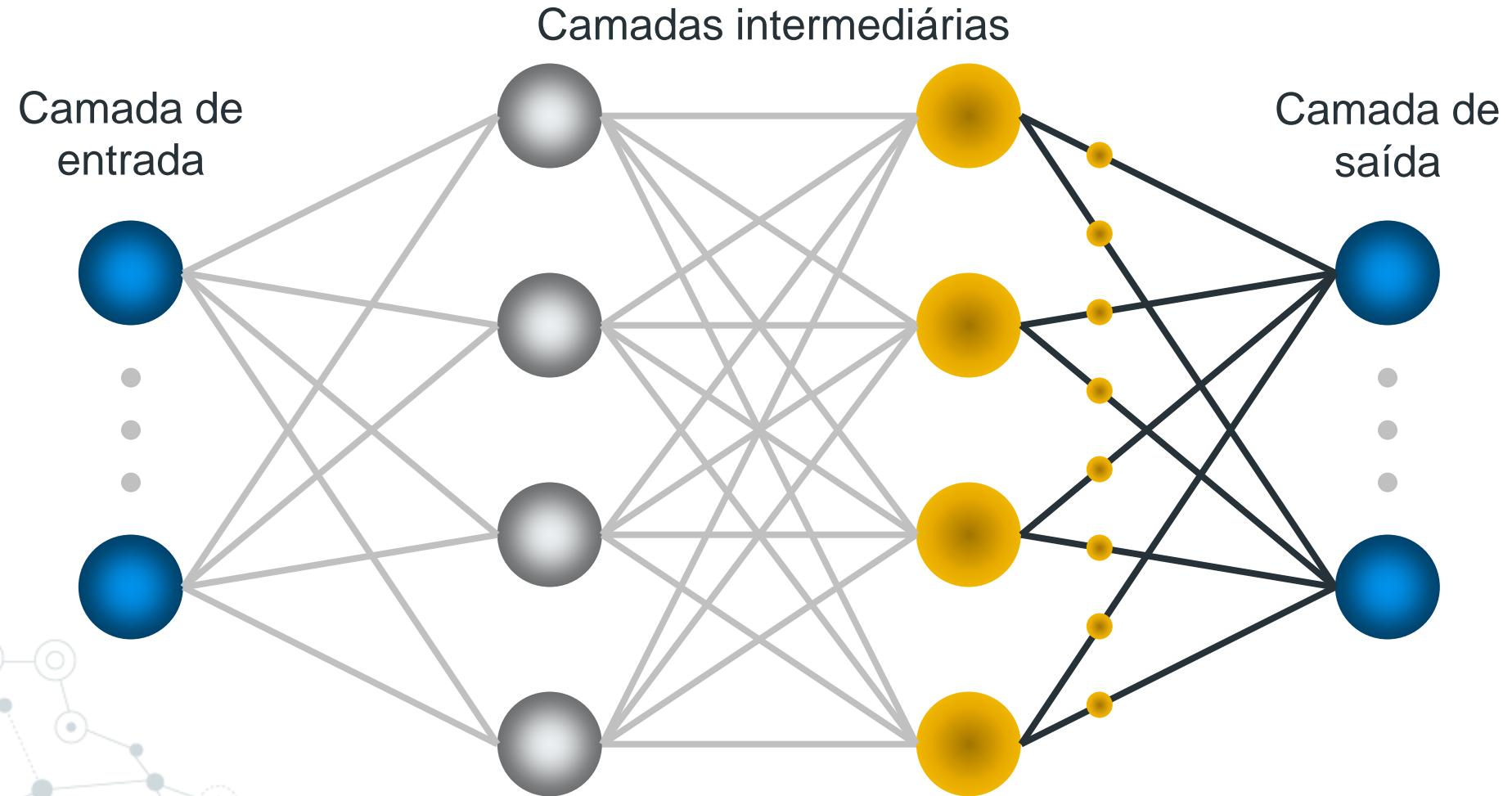
Os neurônios da camada  $i$  calculam seus sinais de saída e propagam à camada  $i + 1$ .



# ALGORITMO BACKPROPAGATION

- FASE *Forward*:

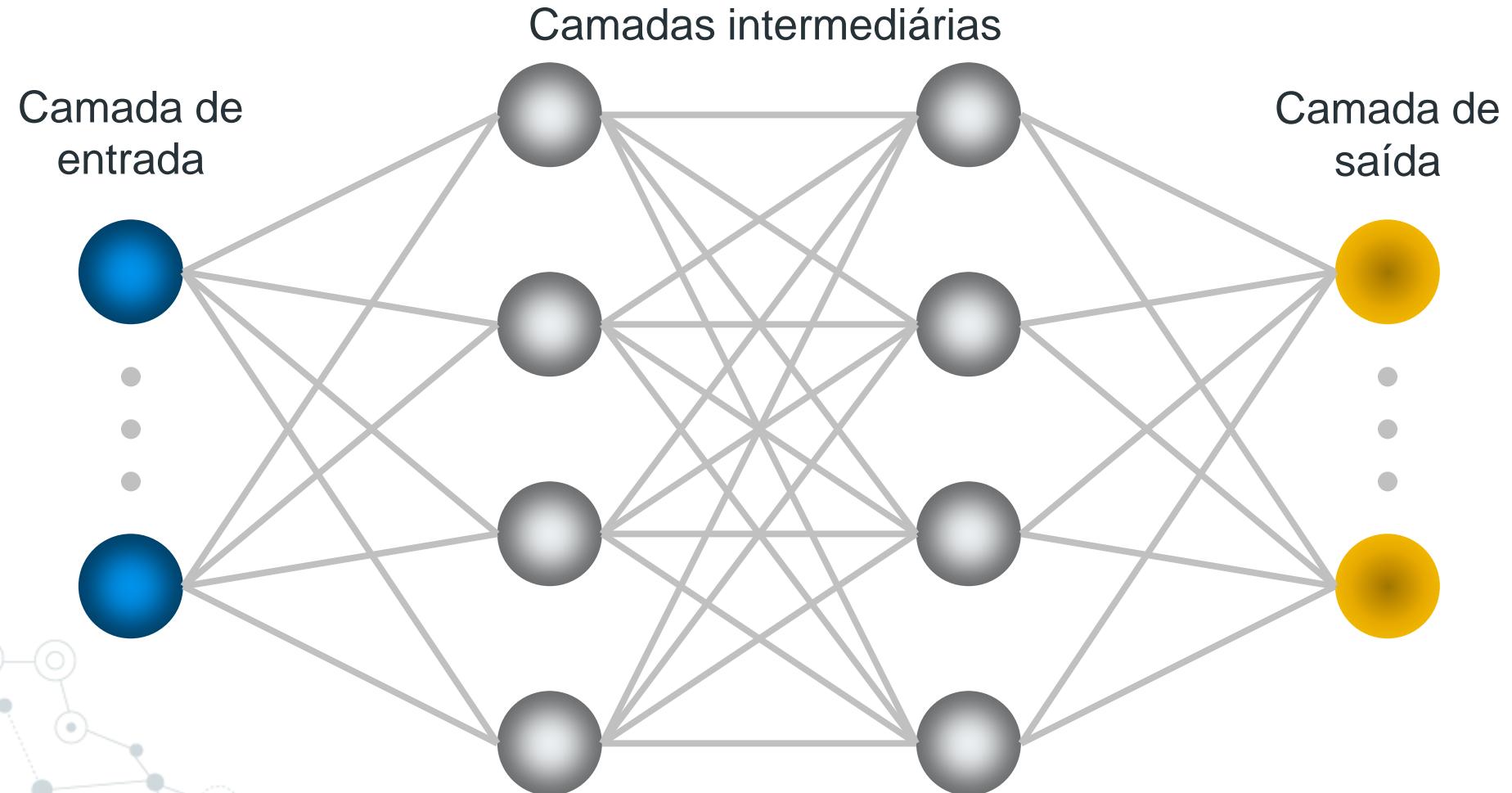
A última camada oculta calcula seus sinais de saída e os envia à camada de saída.



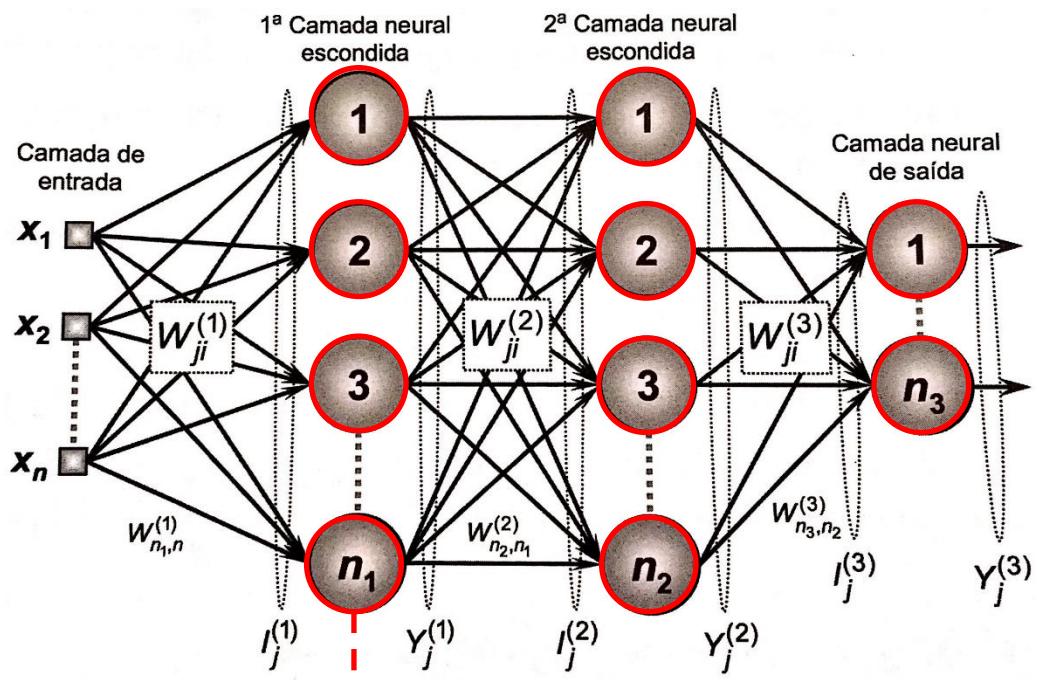
# ALGORITMO BACKPROPAGATION

- FASE *Forward*:

A camada de saída calcula os valores de saída da rede.



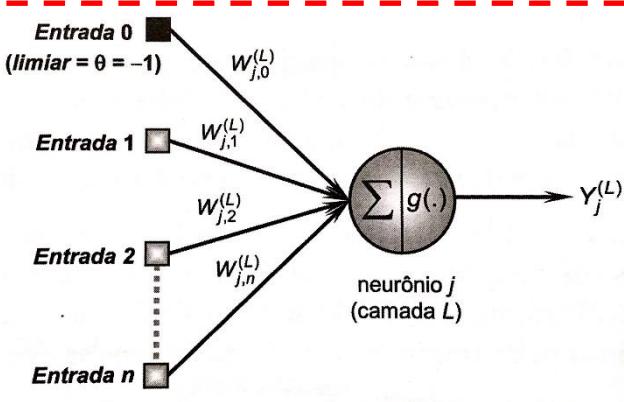
# ALGORITMO BACKPROPAGATION



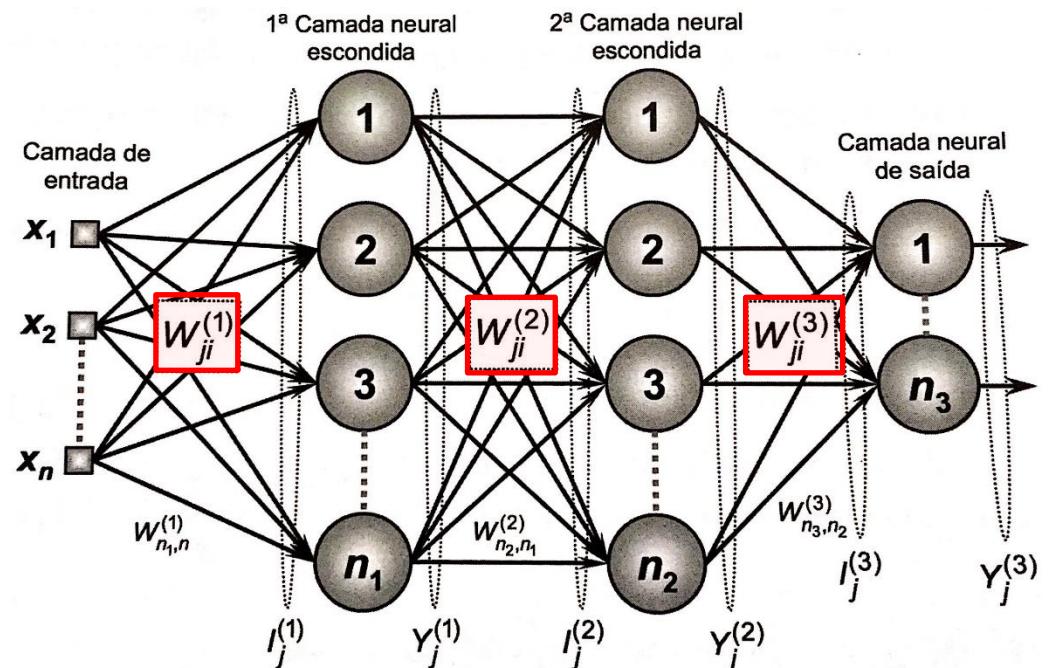
Em cada um dos neurônios  $\{j\}$  pertencentes a uma das camadas  $\{L\}$ , o termo  $g(\cdot)$  representa uma função de ativação que deve ser contínua e diferenciável em todo o seu domínio.

## Exemplos:

- **Tangente hiperbólica:**  $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- **Sigmóide:**  $g(z) = \frac{1}{1 + e^{-z}}$



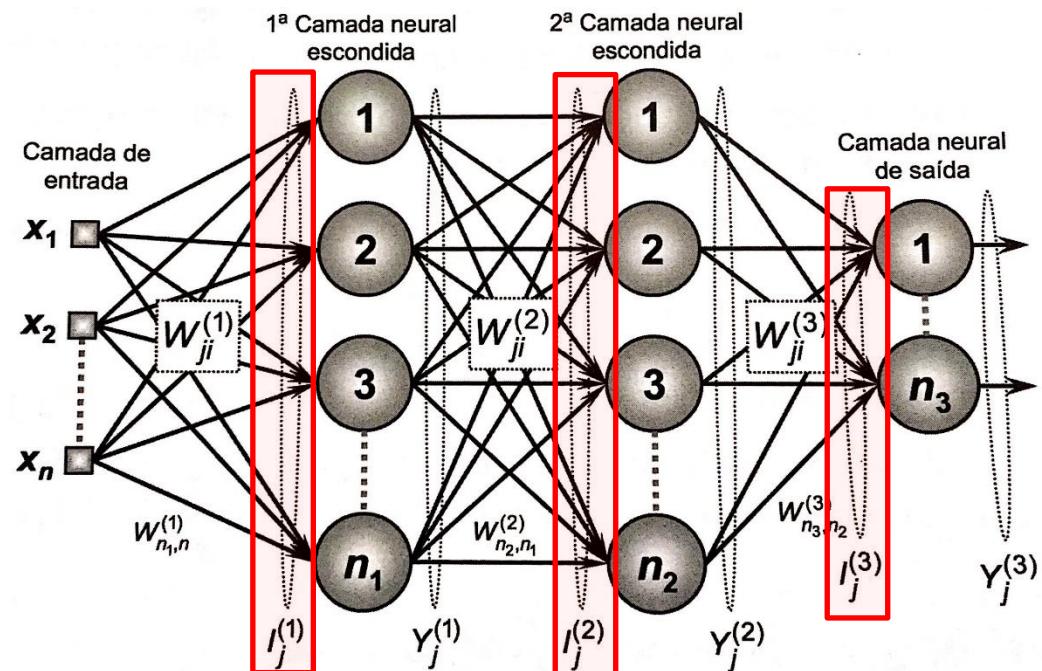
# ALGORITMO BACKPROPAGATION



$W_{ji}^{(L)}$  são matrizes de pesos cujos elementos denotam o valor do peso sináptico conectando o  $j$ -ésimo neurônio da camada ( $L$ ) ao  $i$ -ésimo neurônio da camada ( $L - 1$ ).

- $W_{ji}^{(3)}$  é o peso sináptico conectando o  $j$ -ésimo neurônio da camada de saída ao  $i$ -ésimo neurônio da camada escondida 2.
- $W_{ji}^{(2)}$  é o peso sináptico conectando o  $j$ -ésimo neurônio da camada escondida 2 ao  $i$ -ésimo neurônio da camada escondida 1.
- $W_{ji}^{(1)}$  é o peso sináptico conectando o  $j$ -ésimo neurônio da camada escondida 1 ao  $i$ -ésimo sinal da camada de entrada.

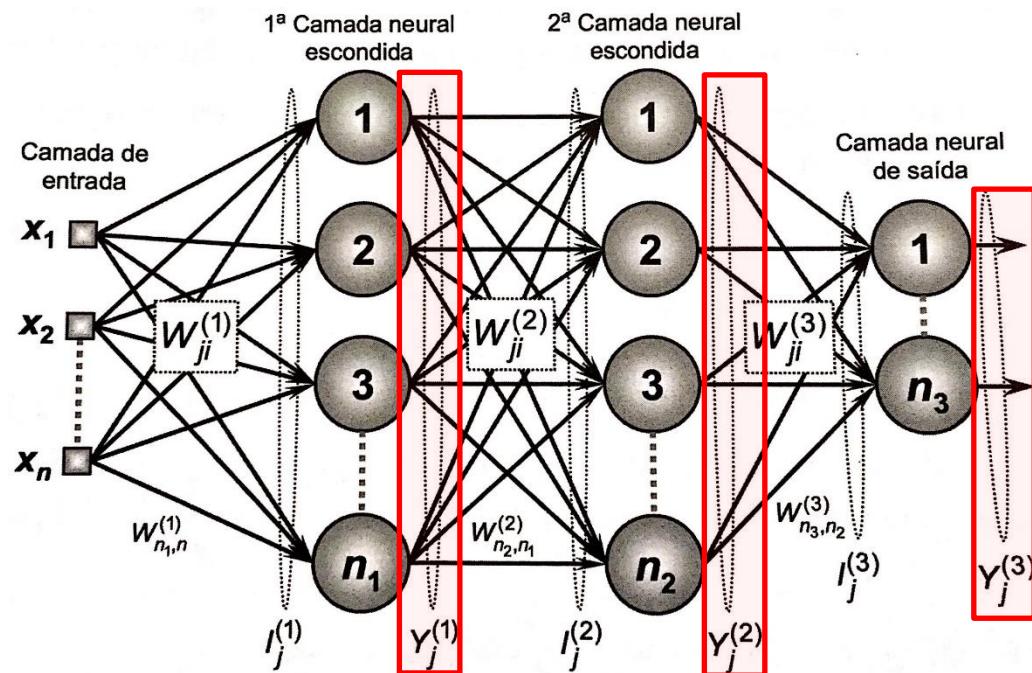
# ALGORITMO BACKPROPAGATION



$I_j^{(L)}$  são vetores cujos elementos denotam a entrada ponderada em relação ao  $j$ -ésimo

- $I_j^{(1)} = \sum_{i=0}^n W_{ji}^{(1)} \cdot x_i \Leftrightarrow I_j^{(1)} = W_{j,0}^{(1)} \cdot x_0 + W_{j,1}^{(1)} \cdot x_1 + \dots + W_{j,n}^{(1)} \cdot x_n$
- $I_j^{(2)} = \sum_{i=0}^n W_{ji}^{(2)} \cdot Y_i^{(2)} \Leftrightarrow I_j^{(2)} = W_{j,0}^{(2)} \cdot Y_0^{(1)} + W_{j,1}^{(2)} \cdot Y_1^{(1)} + \dots + W_{j,n_1}^{(2)} \cdot Y_{n_1}^{(1)}$
- $I_j^{(3)} = \sum_{i=0}^n W_{ji}^{(3)} \cdot Y_i^{(3)} \Leftrightarrow I_j^{(3)} = W_{j,0}^{(3)} \cdot Y_0^{(2)} + W_{j,1}^{(3)} \cdot Y_1^{(2)} + \dots + W_{j,n_2}^{(3)} \cdot Y_{n_2}^{(2)}$

# ALGORITMO BACKPROPAGATION

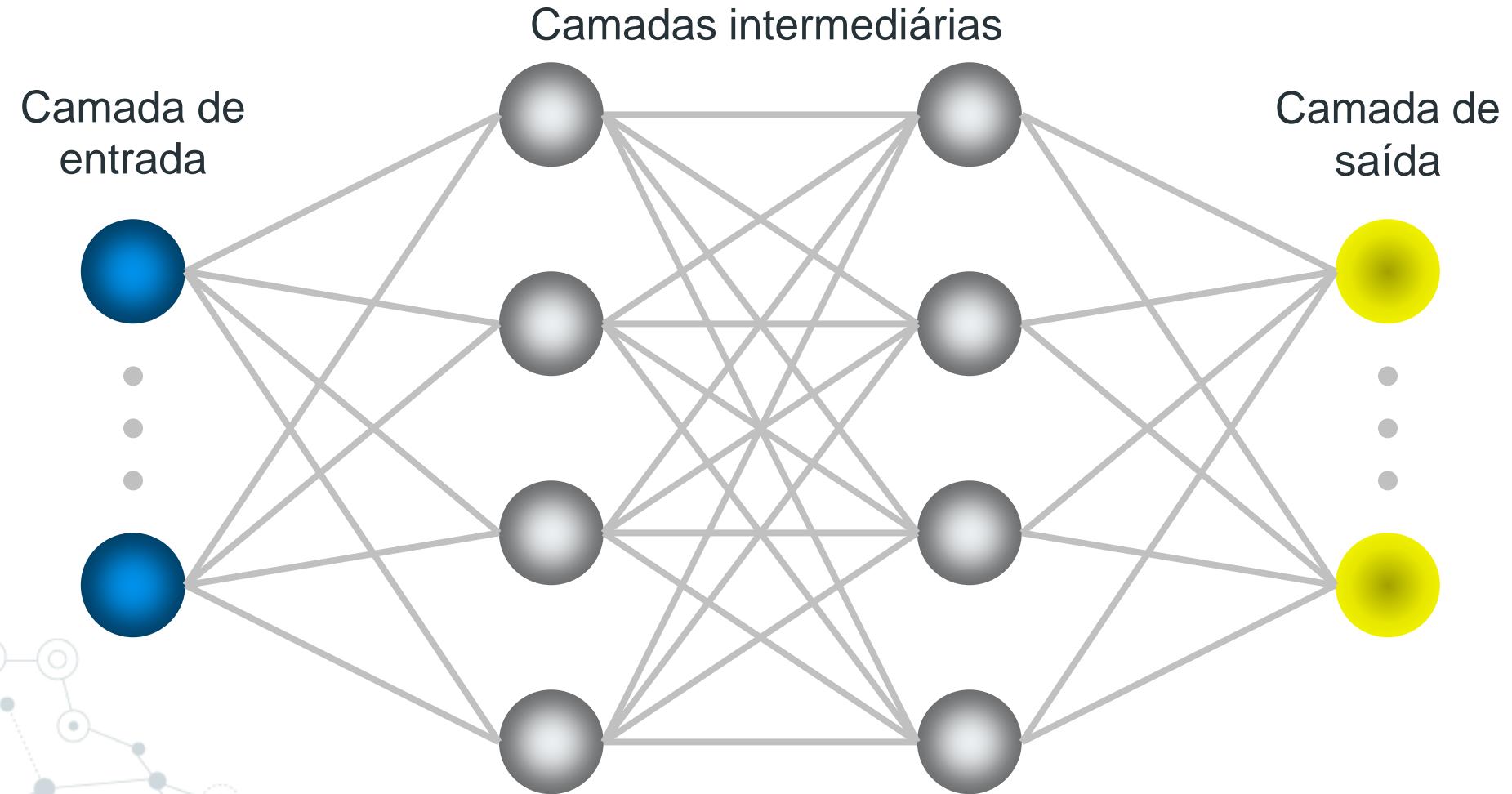


$I_{ji}^{(L)}$  são vetores cujos elementos denotam a saída do  $j$ -ésimo neurônio da camada L, os quais são definidos por:

- $Y_j^{(1)} = g(I_j^{(1)})$
- $Y_j^{(2)} = g(I_j^{(2)})$
- $Y_j^{(3)} = g(I_j^{(3)})$

# ALGORITMO BACKPROPAGATION

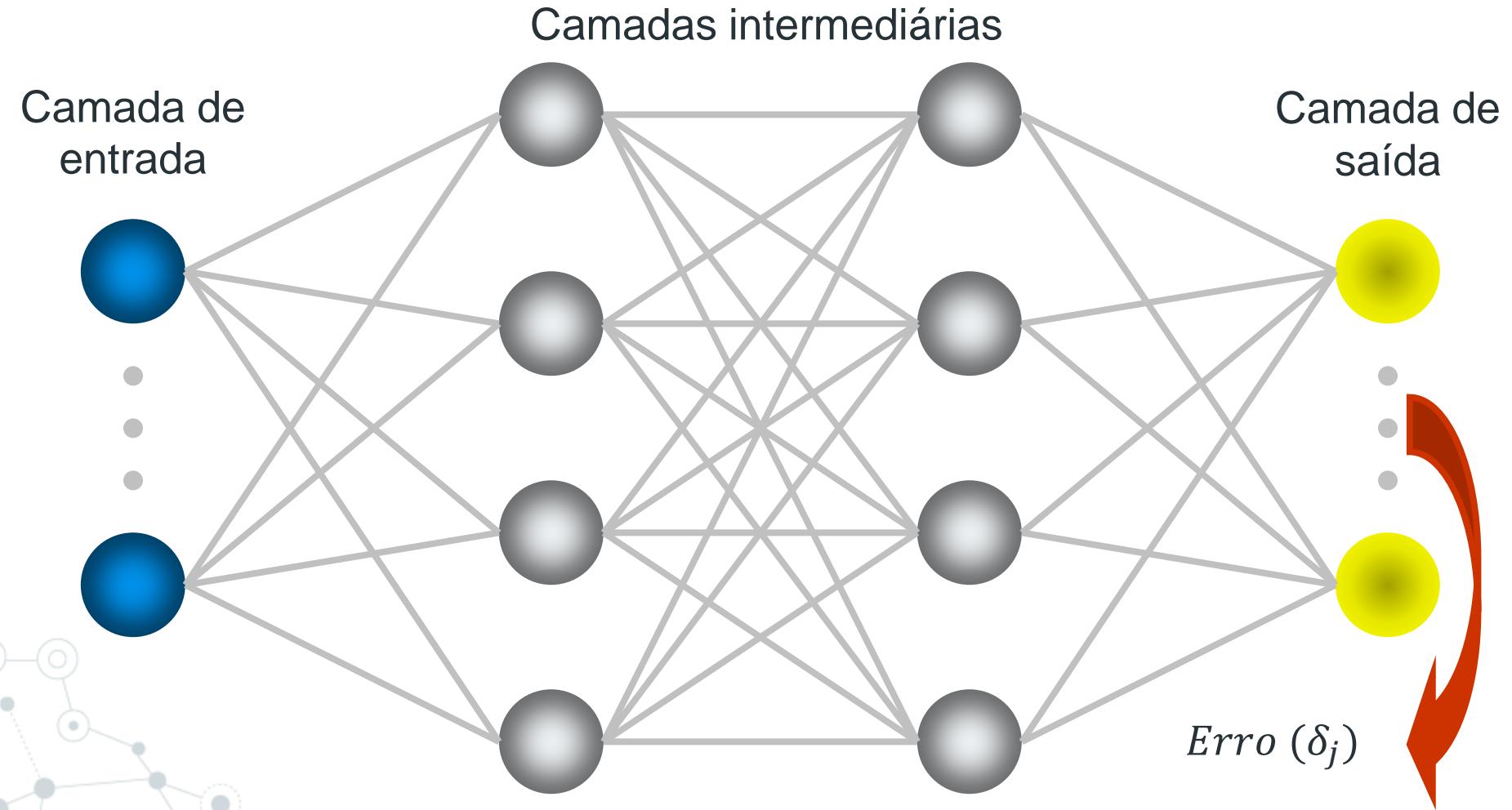
- FASE *Backward*:



# ALGORITMO BACKPROPAGATION

- FASE *Backward*:

A camada de saída calcula o erro da rede  $\delta_j$ .



# ALGORITMO BACKPROPAGATION

- **Ajuste dos Pesos Sinápticos da Camada de Saída:**

- O objetivo do processo de treinamento para a camada neural de saída consiste em ajustar a matriz de pesos  $W_{ji}^{(3)}$  a fim de minimizar o erro entre a saída produzida pela rede em relação à respectiva saída desejada.
- A regra de ajuste é similar àquela aplicada ao Adaline:

$$W_{ji}^{(3)} = W_{ji}^{(3)} + \eta \cdot \delta_j^{(3)} \cdot Y_i^{(2)}$$

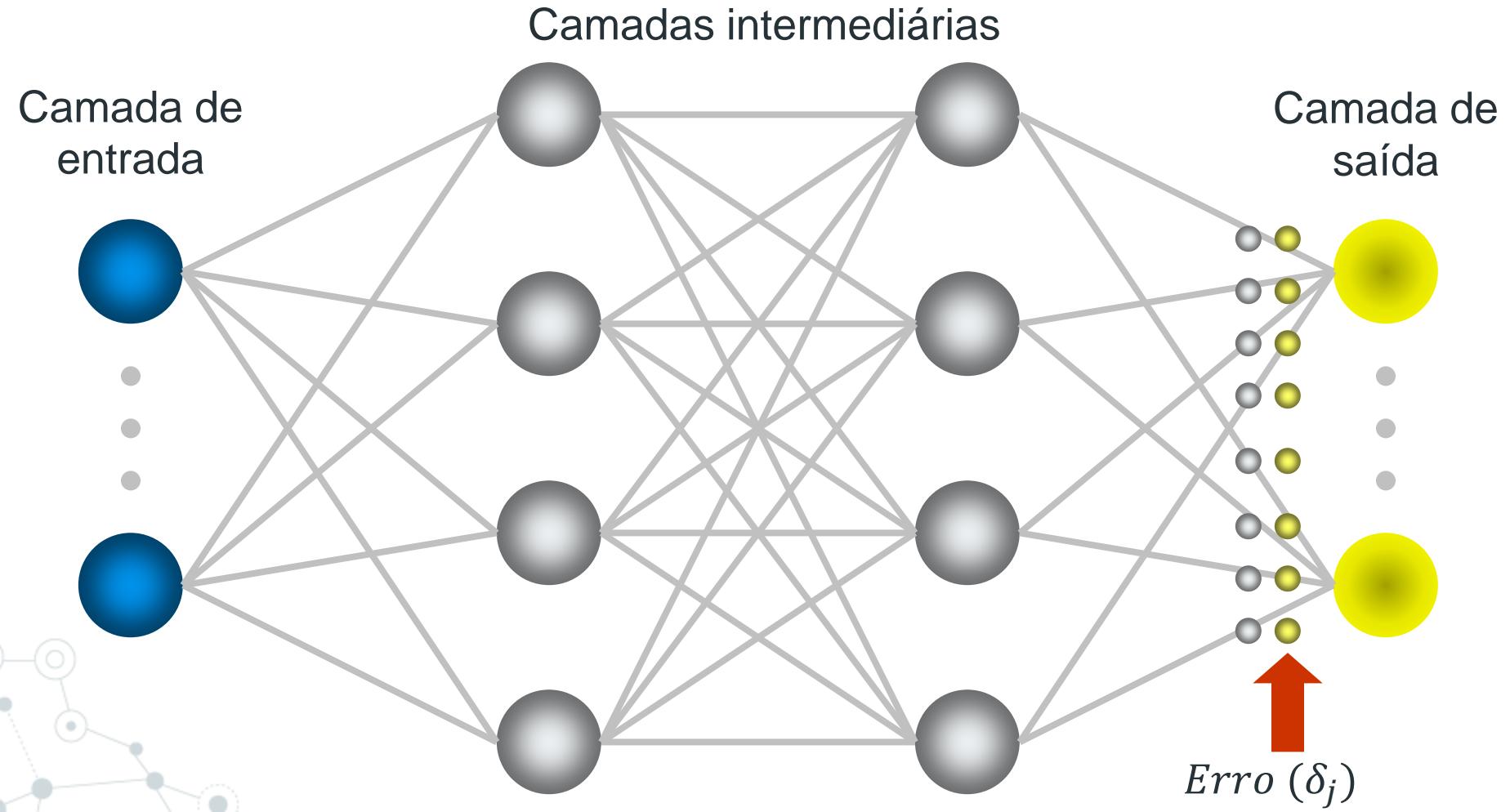
- O termo  $\delta_j^{(3)}$  é definido como o gradiente local em relação ao  $j$ -ésimo neurônio da camada de saída, sendo o mesmo dado por:

$$\delta_j^{(3)} = (d_j - Y_j^{(3)}) \cdot g'(I_j^{(3)})$$

# ALGORITMO BACKPROPAGATION

- FASE *Backward*:

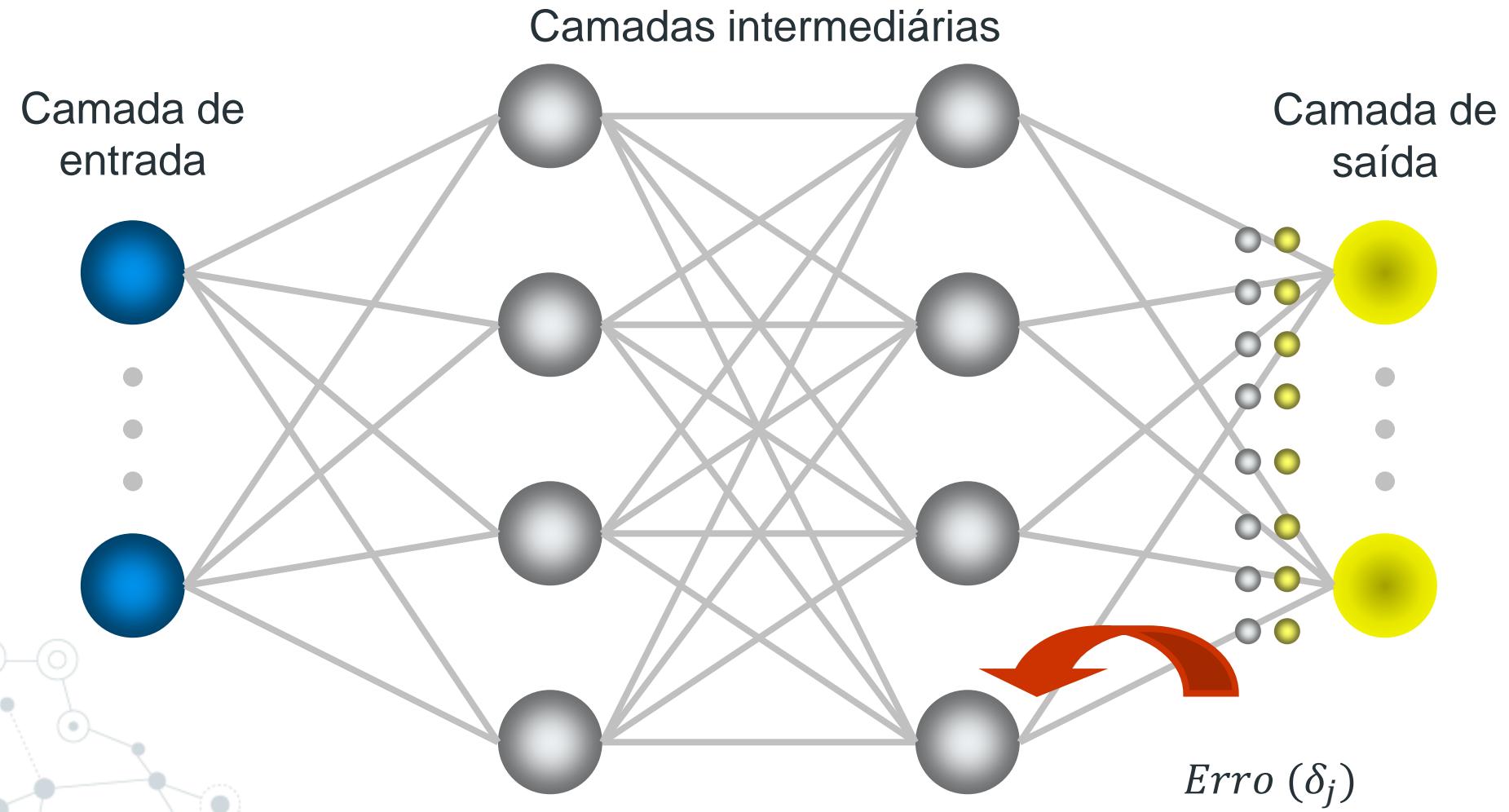
Calcula o termo de correção dos pesos (a atualização será feita depois)  $\Delta w_{ji} = \delta_j Y_i$ .



# ALGORITMO BACKPROPAGATION

- FASE *Backward*:

Propaga o erro para a última camada oculta.



# ALGORITMO BACKPROPAGATION

- **Ajuste dos Pesos Sinápticos de Camadas Intermediárias:**
  - Diferentemente dos neurônios pertencentes à camada de saída do MLP, para os neurônios das camadas intermediárias não se tem acesso de forma direta aos valores desejados para as suas saídas. Nesta situação, os ajustes de seus pesos sinápticos são efetuados por intermédios de estimativas de erros de saídas produzidos por aqueles neurônios da camada imediatamente posterior, os quais já foram ajustados.
  - Assim, é justamente neste aspecto que se encontra **a essência do algoritmo de retropropagação do erro *backpropagation*, pois:**
    - Em primeira instância, tem-se ajustado os pesos sinápticos dos neurônios da camada de saída mediante valores verdadeiros dos desvios observados entre suas respostas produzidas e os respectivos valores desejados.
    - Em segunda instância, retropropaga-se este erro para os neurônios das camadas anteriores, ponderando-se os mesmos pelos valores de pesos sinápticos que já foram previamente ajustados em todas as camadas posteriores.

Consequentemente, a resposta desejada para um neurônio de camada escondida deve ser então determinada em função dos neurônios (da camada imediatamente posterior) que estão diretamente conectados a este e que já foram previamente ajustados no passo anterior.

# ALGORITMO BACKPROPAGATION

- **Ajuste dos Pesos Sinápticos da Segunda Camadas Escondida:**
  - O objetivo do processo de treinamento para a segunda camada neural consistem em ajustas a matriz de pesos  $W_{ji}^{(2)}$  a fim de minimizar o erro entre a saída produzida pela rede em relação à retropropagação do erro advindo dos ajustes dos neurônios da camada neural de saída.

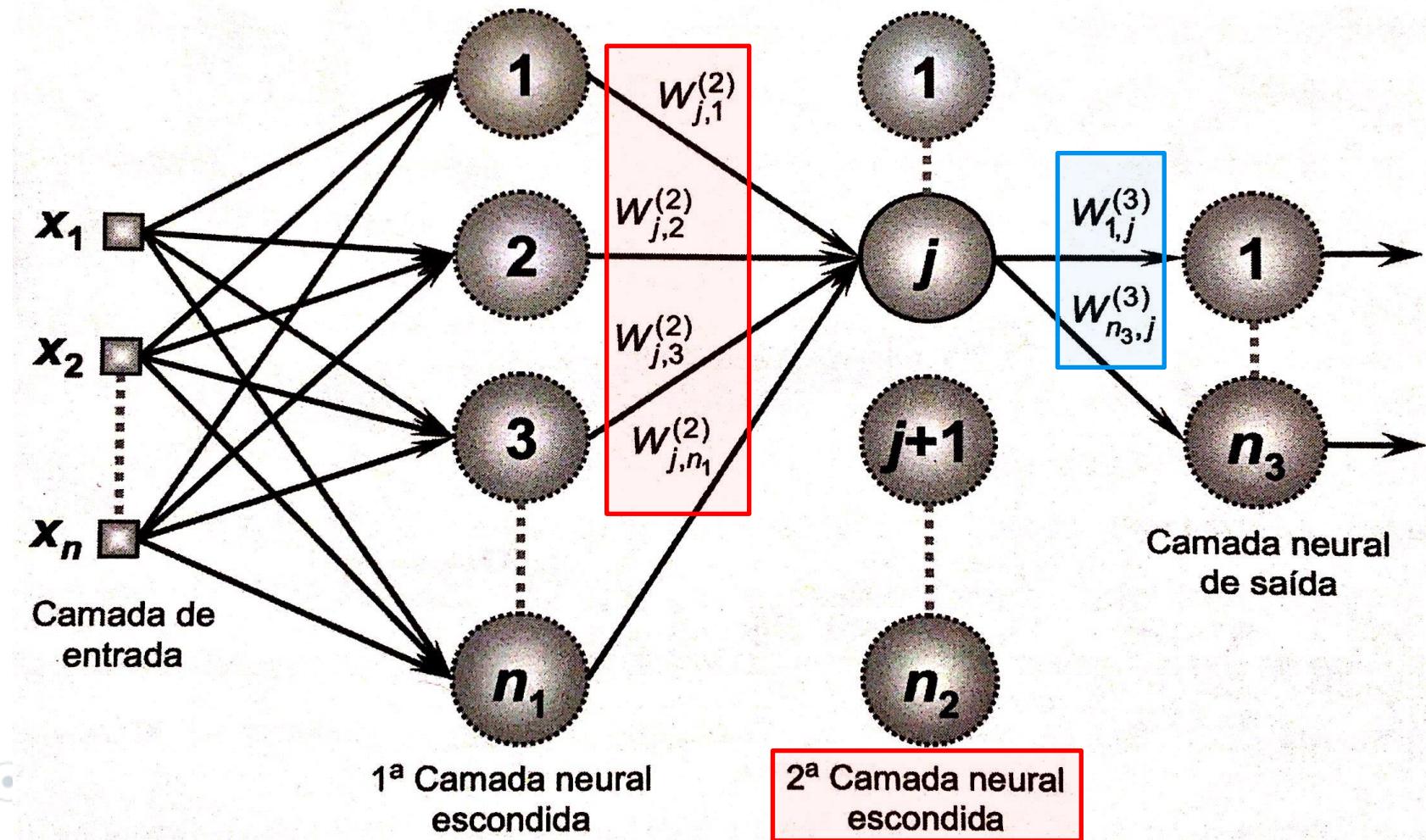
$$W_{ji}^{(2)} = W_{ji}^{(2)} + \eta \cdot \delta_j^{(2)} \cdot Y_i^{(1)}$$

- O termo  $\delta_j^{(2)}$  é definido como o gradiente local em relação ao  $j$ -ésimo neurônio da segunda camada intermediária, isto é:

$$\delta_j^{(2)} = \sum_{k=1}^{n_3} (\delta_k^{(3)} \cdot W_{kj}^{(3)}) \cdot g'(I_j^{(2)})$$

- Ressalta-se aqui novamente a essência do método backpropagation, pois todos os pesos  $W_{ij}^{(3)}$  já foram ajustados no passo anterior com base em valores reais de erro, sendo que estes serão então utilizados para o ajuste dos pesos da segunda camada neural intermediária.

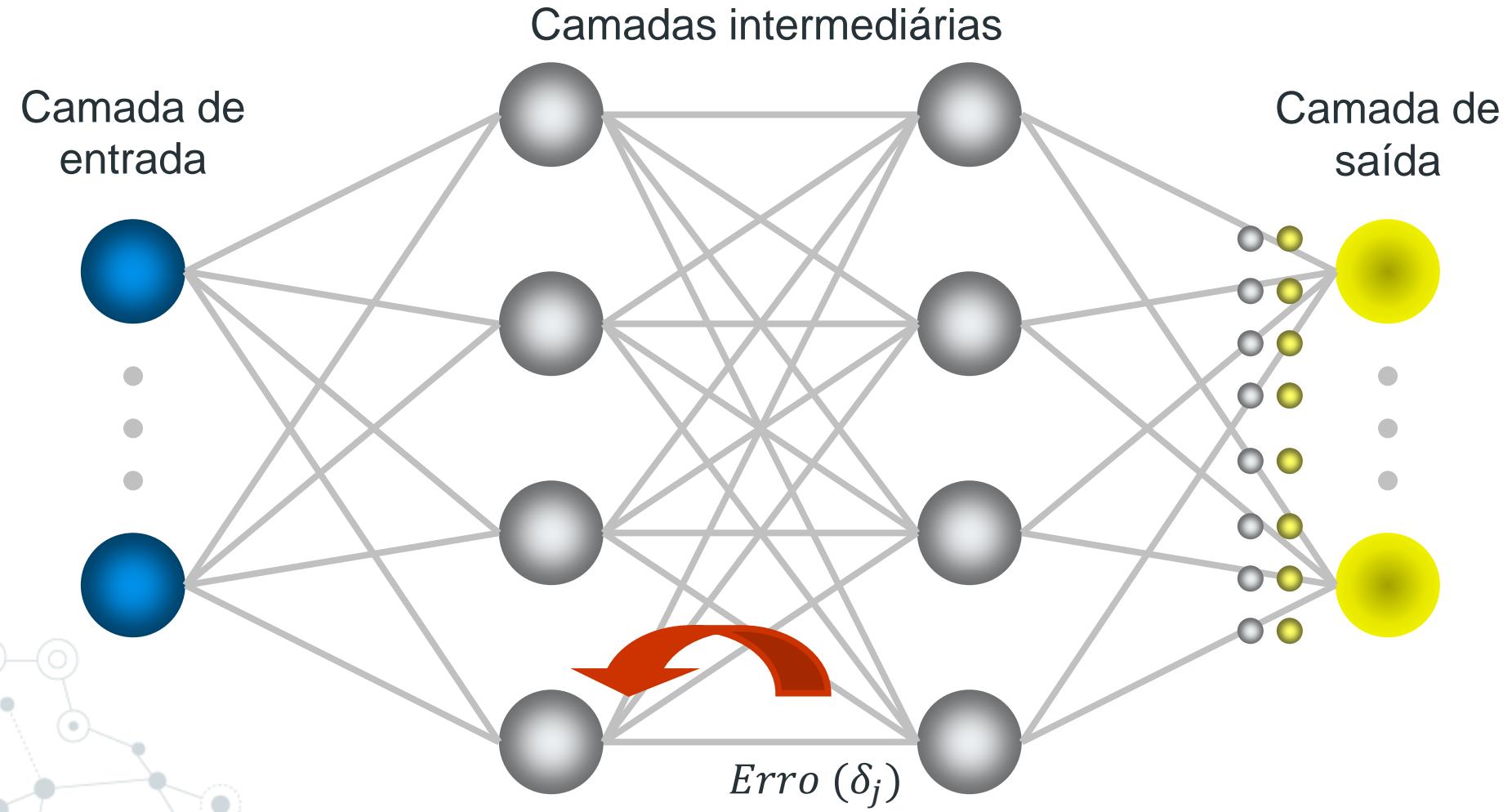
# ALGORITMO BACKPROPAGATION



# ALGORITMO BACKPROPAGATION

- FASE *Backward*:

Propaga o erro para a primeira camada oculta.



# ALGORITMO BACKPROPAGATION

## ○ Ajuste dos Pesos Sinápticos da Primeira Camadas Escondida:

- Em relação à primeira camada escondida, o objetivo do processo de treinamento consiste em ajustar a matriz de pesos  $W_{ji}^{(1)}$  a fim de minimizar o erro entre a saída produzida pela rede em função da retropropagação do erro advindo dos ajustes dos neurônios da segunda camada escondida.

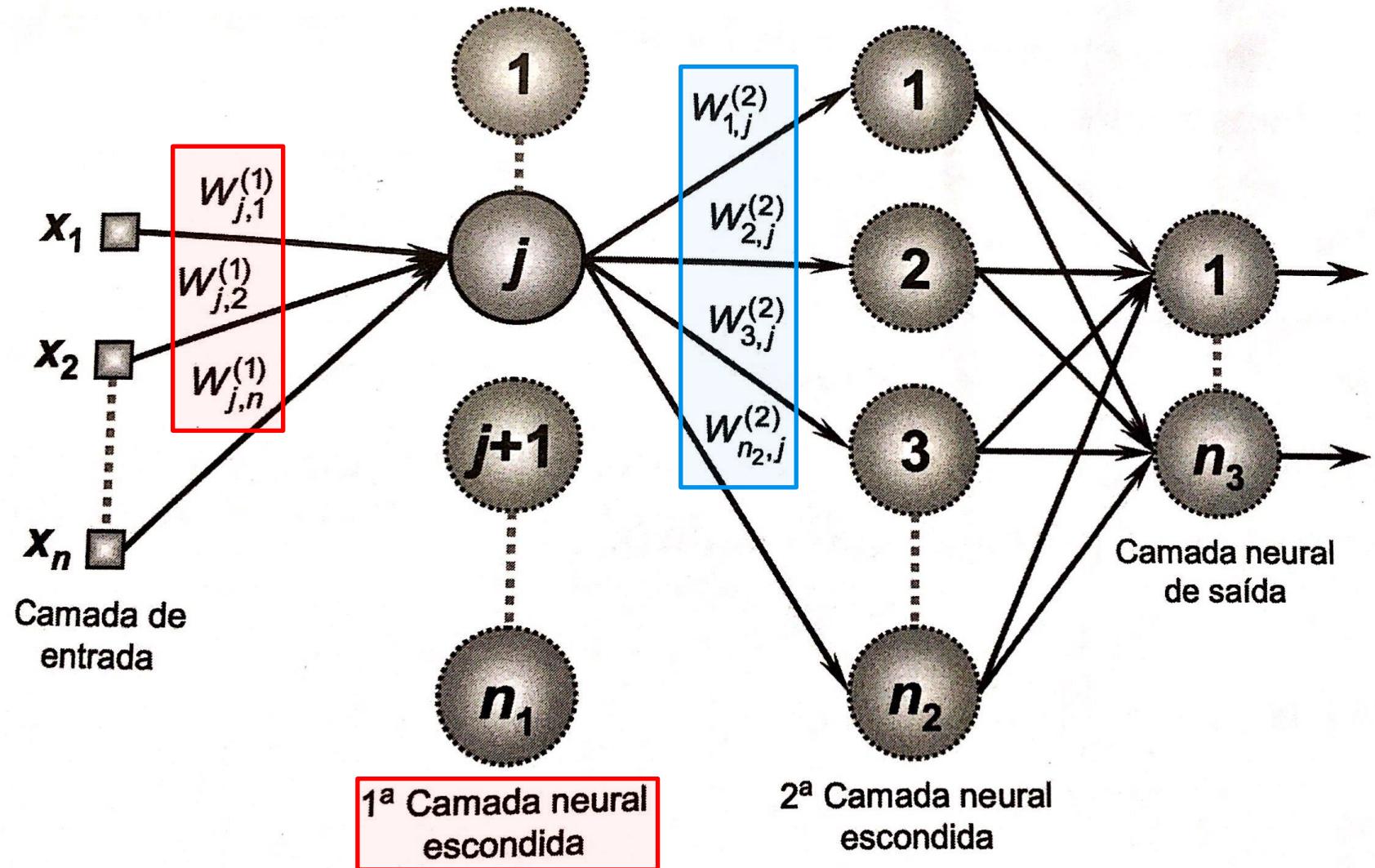
$$W_{ji}^{(1)} = W_{ji}^{(1)} + \eta \cdot \delta_j^{(1)} \cdot x_i$$

- O termo  $\delta_j^{(1)}$  é definido como o gradiente local em relação ao  $j$ -ésimo neurônio da primeira camada intermediária, isto é:

$$\delta_j^{(1)} = \sum_{k=1}^{n_2} (\delta_k^{(2)} \cdot W_{kj}^{(2)}) \cdot g'(I_j^{(1)})$$

Portanto, a equação acima ajusta os pesos dos neurônios da primeira camada neural intermediária, levando-se em consideração a retropropagação do erro advinda a partir dos neurônios da segunda camada intermediária.

# ALGORITMO BACKPROPAGATION



# ALGORITMO BACKPROPAGATION

## Calculo de $\varphi'(n)$

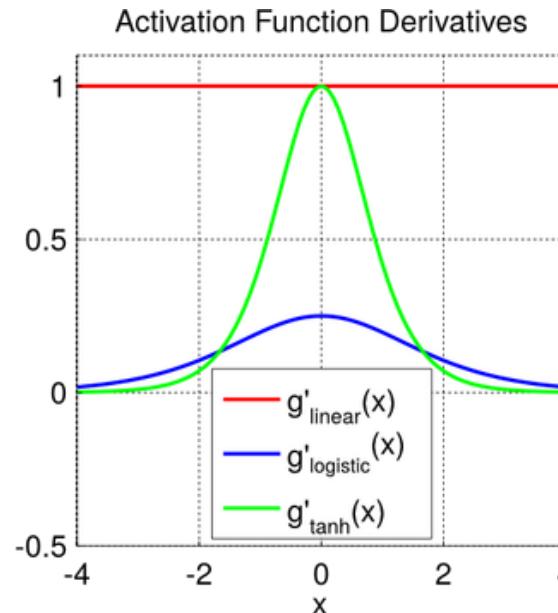
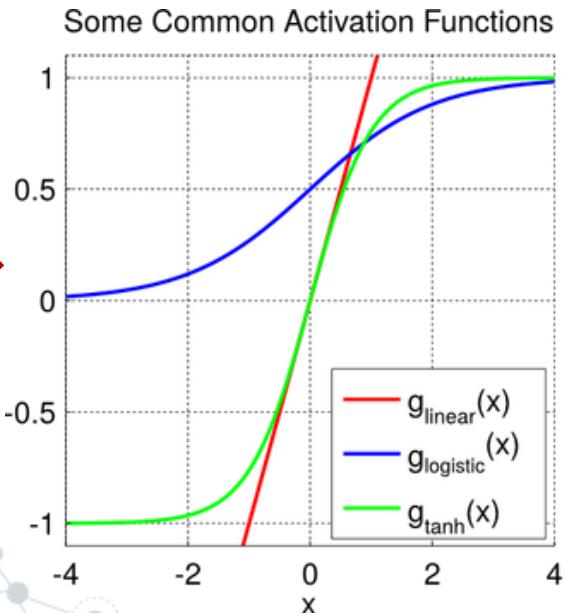
- Função de ativação do tipo tangente hiperbólica:

$$g(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g(z) = g(z)(1 - g(z))$$

- Função de ativação do tipo sigmoide:

$$g(z) = \frac{1}{1 + e^{-z}}$$



$$g'(z) = 1 - \frac{\sinh^2(z)}{\cosh^2(z)}$$
$$g'(z) = 1 - g^2(z)$$

# ALGORITMO BACKPROPAGATION

- **Faror de Aprendizagem  $\eta$ :**

- Normalmente  $0 < \eta < 1$ .
- Pode ter um valor diferente para cada camada de pesos sinápticos.
- Deve ter um valor maior nas camadas mais perto da entrada.
- Há técnicas para redução progressiva da taxa de aprendizado (decaimento linear).

# ALGORITMO MLP - TREINAMENTO

Início {Algoritmo PMC – Fase de Treinamento}

- <1> Obter o conjunto de amostras de treinamento { $x^{(k)}$ };
- <2> Associar o vetor de saída desejada { $d^{(k)}$ } para cada amostra;
- <3> Iniciar  $w_{ji}^{(1)}$ ,  $w_{ji}^{(2)}$  e  $w_{ji}^{(3)}$  com valores aleatórios pequenos;
- <4> Especificar taxa de aprendizagem { $\eta$ } e precisão requerida { $\varepsilon$ };
- <5> Iniciar o contador de número de épocas {época  $\leftarrow 0$ };
- <6> Repetir as instruções:
  - <6.1>  $E_M^{\text{anterior}} \leftarrow E_M$ ; {conforme (5.8)}
  - <6.2> Para todas as amostras de treinamento { $x^{(k)}, d^{(k)}$ }, fazer:
    - <6.2.1> Obter  $I_j^{(1)}$  e  $Y_j^{(1)}$ ;
    - <6.2.2> Obter  $I_j^{(2)}$  e  $Y_j^{(2)}$ ;
    - <6.2.3> Obter  $I_j^{(3)}$  e  $Y_j^{(3)}$ ;
    - <6.2.4> Determinar  $\delta_j^{(3)}$ ;
    - <6.2.5> Ajustar  $w_{ji}^{(3)}$ ;
    - <6.2.6> Determinar  $\delta_j^{(2)}$ ;
    - <6.2.7> Ajustar  $w_{ji}^{(2)}$ ;
    - <6.2.8> Determinar  $\delta_j^{(1)}$ ;
    - <6.2.9> Ajustar  $w_{ji}^{(1)}$ ;
  - <6.3> Obter  $Y_j^{(3)}$  ajustado; {conforme <6.2.1>, <6.2.2> e <6.2.3>}
  - <6.4>  $E_M^{\text{atual}} \leftarrow E_M$ ; {conforme (5.8)}
  - <6.5> época  $\leftarrow$  época + 1;

Até que:  $|E_M^{\text{atual}} - E_M^{\text{anterior}}| \leq \varepsilon$

Fim {Algoritmo PMC – Fase de Treinamento}

# ALGORITMO MLP - OPERAÇÃO

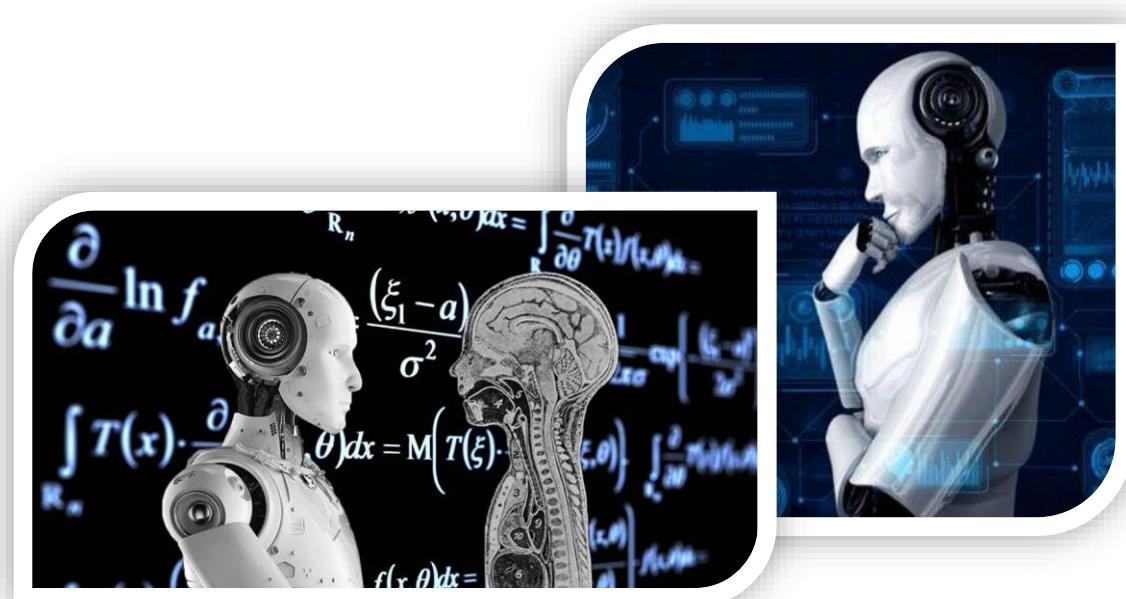
**Início {Algoritmo PMC – Fase de Operação}**

- {<1> Obter uma amostra {  $x$  };
- <2> Assumir  $W_{ji}^{(1)}$ ,  $W_{ji}^{(2)}$  e  $W_{ji}^{(3)}$  já ajustadas no treinamento;
- <3> Execute as seguintes instruções:
  - <3.1> Obter  $I_j^{(1)}$  e  $Y_j^{(1)}$ ;
  - <3.2> Obter  $I_j^{(2)}$  e  $Y_j^{(2)}$ ;
  - <3.3> Obter  $I_j^{(3)}$  e  $Y_j^{(3)}$ ;
- <4> Disponibilizar as saídas da rede, as quais são dadas pelos elementos contidos em  $Y_j^{(3)}$

**Fim {Algoritmo PMC – Fase de Operação}**

# REDES PERCEPTRON MULTICAMADAS

- Versões Aprimoradas do Algoritmo Backpropagation.



# ALGORITMO BACKPROPAGATION

## Momentum:

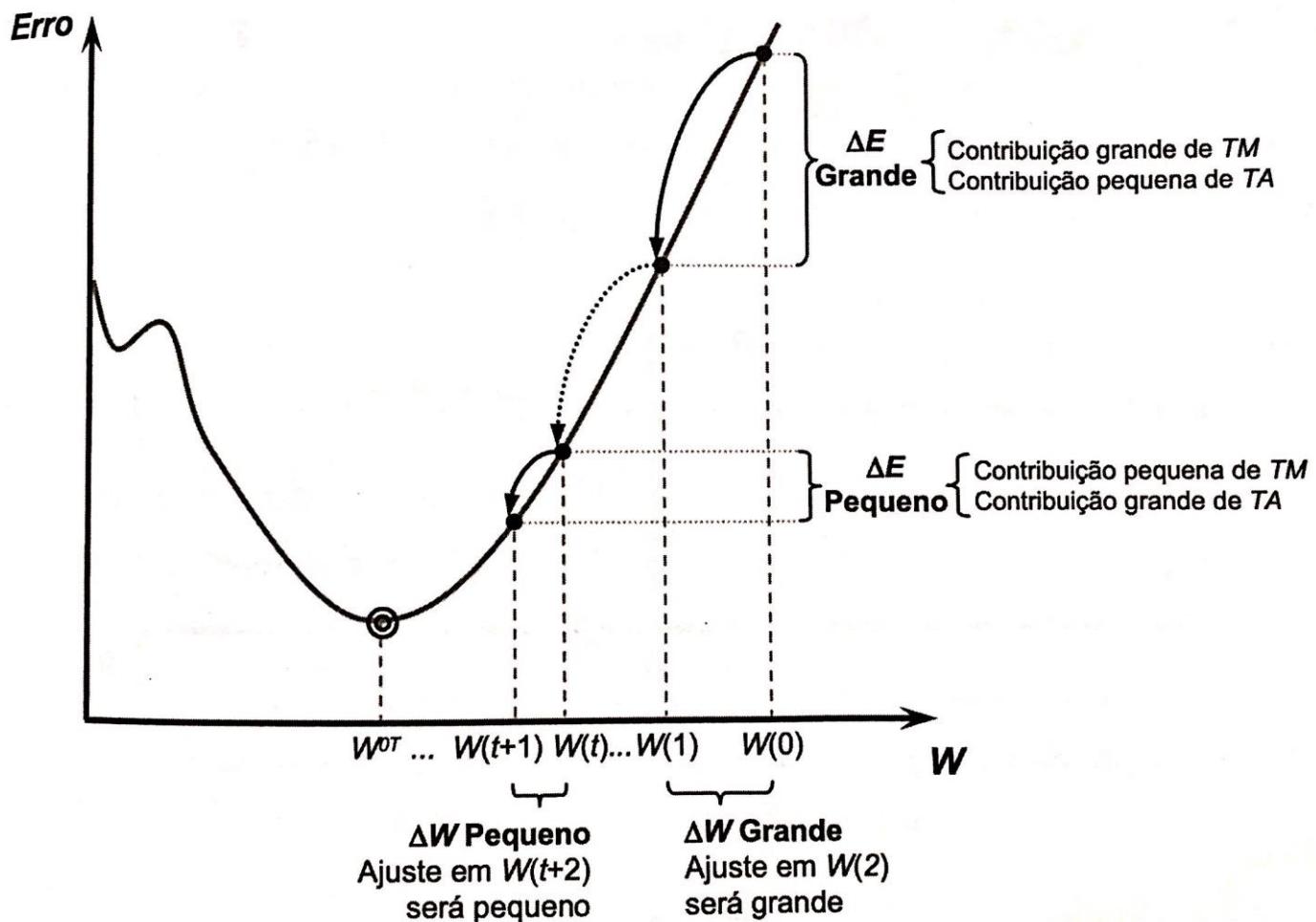
- O *momentum* ( $0 < \alpha < 0.9$ ) permite a travessia de platôs com mais facilidade devido à “inércia”.

$$W_{ji}^{(L)}(t+1) = W_{ji}^{(L)}(t) + \underbrace{\alpha \cdot (W_{ji}^{(L)}(t) - W_{ji}^{(L)}(t-1))}_{\text{termo de momentum}} + \underbrace{\eta \cdot \delta_j^{(L)} \cdot Y_i^{(L-1)}}_{\text{termo de aprendizagem}}$$

- Quando a solução atual estiver **longe** da final, a variação na direção oposta ao gradiente do EQM de iterações sucessivas será grande  
→ grande contribuição do momentum
- Quando a solução atual estiver **próxima** da solução final, as variações nas matrizes de pesos serão então bem ínfimas, pois o EQM entre duas iterações sucessivas será baixa  
→ baixa contribuição do momentum

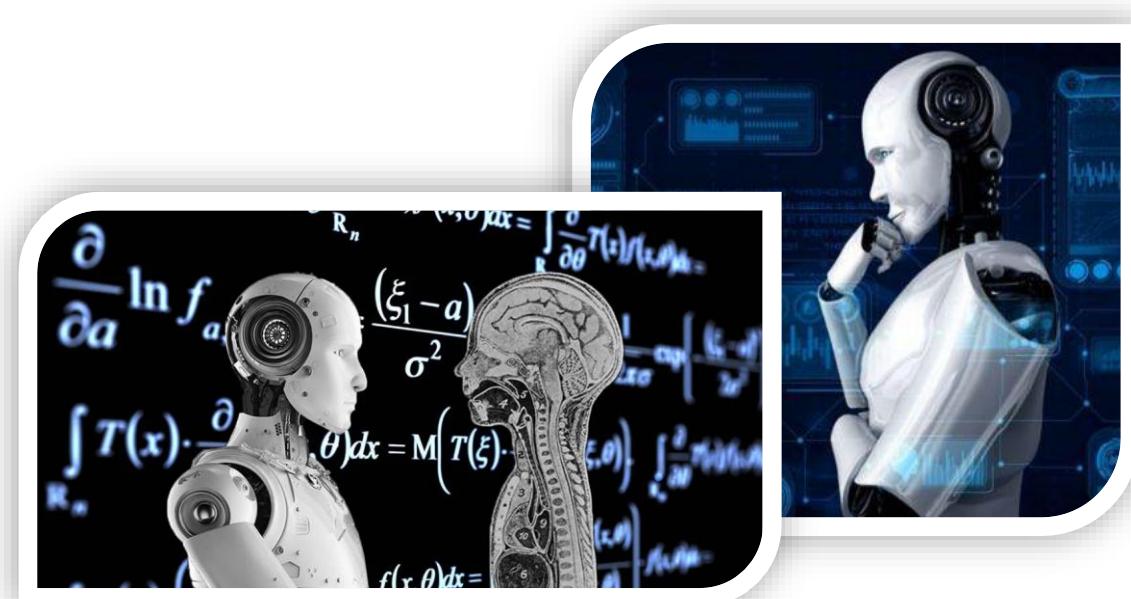
# ALGORITMO BACKPROPAGATION

## Momentum:



# REDES PERCEPTRON MULTICAMADAS

- Métodos de treinamento, validação e teste, validação cruzada, Underfitting e Overfitting.



# PERCEPTRON MULTICAMADAS

## ○ Apresentação dos Exemplos para Aprendizado:

- A ordem de apresentação dos dados durante o aprendizado pode influenciar o desempenho da rede.
- Uma forma de evitar isto é apresentá-los de forma aleatória.
- **O conjunto de dados deve ser dividido em duas partes:**
  - Aprendizado;
  - Teste (validação e testes finais),

Se o número total de dados é pequeno, **deve-se privilegiar o conjunto de aprendizado.**

# PERCEPTRON MULTICAMADAS

- **Época de Aprendizado**

- Sejam  $(x_k(n), d_k(n))$ ,  $k=1, 2, \dots, N$ , os pares estímulo-resposta para o aprendizado de uma rede neural.
- Uma época de aprendizado se completa quando apresentamos todos os  $N$  pares estímulo-resposta ao algoritmo de treinamento.

# PERCEPTRON MULTICAMADAS

- **Atualização de Pesos Sinápticos**

- **Atualização instantânea:** calcula-se  $\Delta w_{ji}$  e aplica-se a correção a  $w_{ji}$  a cada par apresentado.
- **Atualização por lote:** calcula-se os  $\Delta w_{ji}$  para todos os pares, e aplica-se a correção  $\sum \Delta w_{ji}$  a  $w_{ji}$ , depois que todos os pares foram apresentados.

# PERCEPTRON MULTICAMADAS

- **Observações:**

- Na atualização instantânea, a estimativa do gradiente não é muito boa, e os passos são dados de forma mais aleatória, enquanto que na atualização por lote temos uma aproximação mais precisa para o gradiente.
- Entretanto, na prática, o primeiro método apresenta resultados melhores (menor probabilidade de parar em mínimos locais).

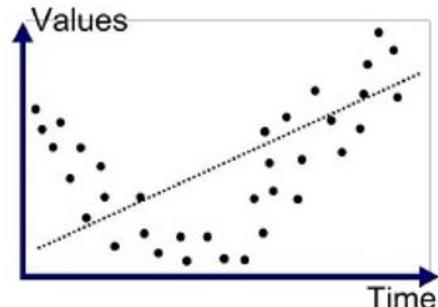
# PERCEPTRON MULTICAMADAS

## Underfitting e Overfitting:

- Estes erros são relativos à razão entre o número de neurônios e o número de dados de treinamento.

### → Overfitting

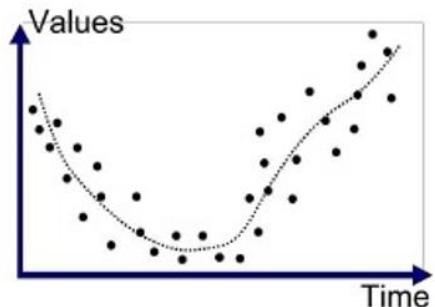
- Erros pequenos no treinamento, mas erros grandes no teste



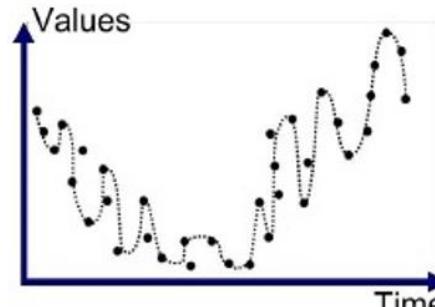
Underfitted

### → Underfitting

- Erros grandes no treinamento e no teste



Good Fit/Robust



Overfitted

# PERCEPTRON MULTICAMADAS

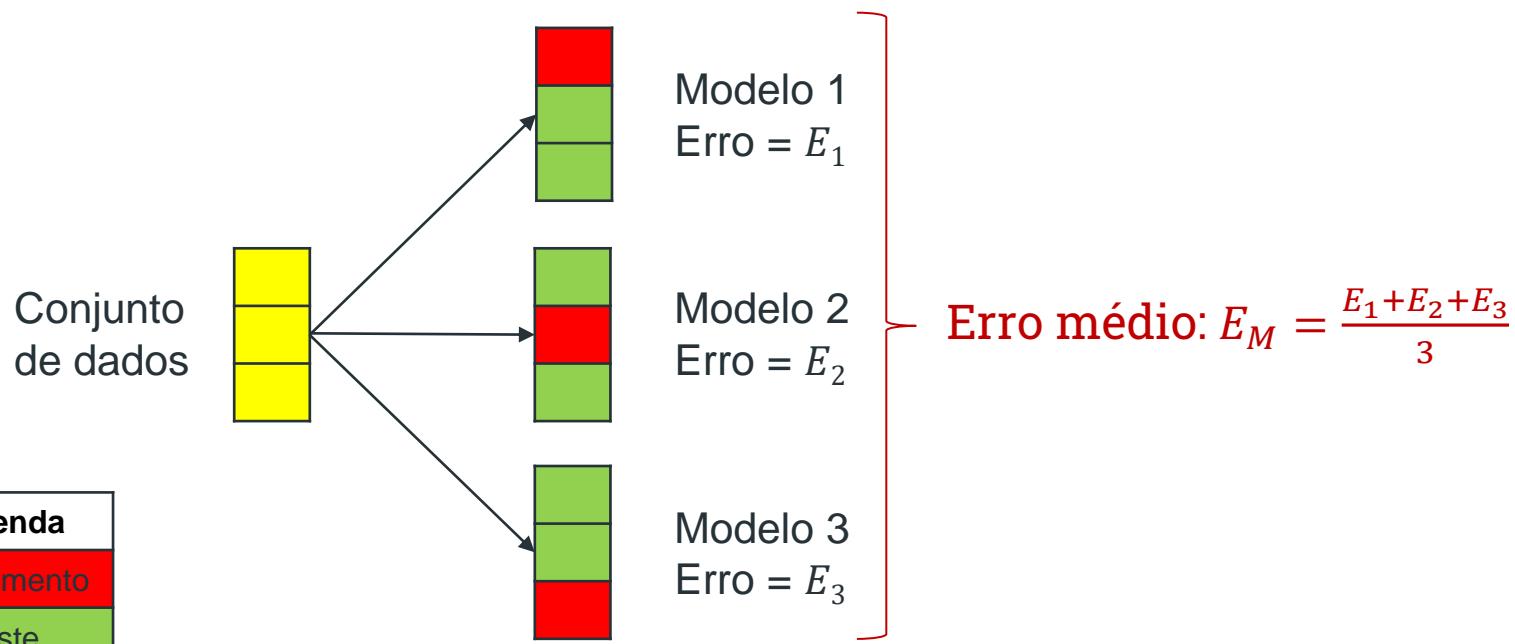
- **Conjunto de Validação:**

- Consiste em fazer a verificação da convergência da rede através de um conjunto de dados diferentes daqueles usados para o aprendizado.
- Tenta evitar que a rede fique viciada nos dados de aprendizado.
- **A base passa a ser dividida em 3 partes:**
  - ✓ pares de aprendizado
  - ✓ pares de validação
  - ✓ pares de teste

# PERCEPTRON MULTICAMADAS

## Validação Cruzada:

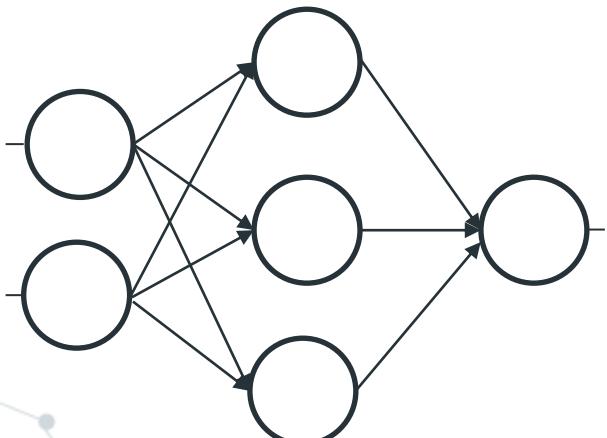
- Técnica estatística para avaliar a capacidade de generalização de um modelo.
- Consiste no particionamento do conjunto de dados em subconjuntos mutualmente exclusivos, e posteriormente, utilizando alguns destes subconjuntos para o treinamento e os demais subconjuntos para teste do modelo.



# PERCEPTRON MULTICAMADAS

## Heurística para Topologia:

- Sabe-se que não existe uma regra ótima para determinar a topologia correta de uma rede neural.
- Entretanto, pode-se utilizar de métodos heurísticos para estabelecer sua configuração inicial.
- Exemplo de heurística: “a quantidade de pesos sinápticos na rede neural deve ser igual ao número de amostras no conjunto de dados”.
- Exemplo do uso desta heurística para um MLP de 1 camada oculta:



**Considerando:**

- $I$  entradas
- $H$  neurônios na camada escondida
- $O$  neurônios na camada de saída
- $T$  amostras no conjunto de dados

**$H$  pode ser estimado da seguinte forma:**

$$T = I \times H + H \times O = (I + O) \times H$$

$$H = \frac{T}{(I + O)}$$

Obrigada pela Atenção!

Dúvidas?

[victoria.souto@Inatel.br](mailto:victoria.souto@Inatel.br)